Federal University of Paraná

Wesley Klewerton Guez Assunção

ModelVars2SPL: An Automated Approach to Reengineer Model Variants into Software Product Lines

Curitiba - PR
2017

Wesley Klewerton Guez Assunção

ModelVars2SPL: An Automated Approach to Reengineer Model Variants into Software Product Lines

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor in Informatics in the Graduate Program in Computer Science, Department of Informatics, Federal University of Paraná.

Field: *Computer Science*.

Advisor: Prof. Dr. Silvia R. Vergilio.

Co-Advisor: Prof. Dr. Roberto E. Lopez-Herrejon.

Curitiba - PR
2017

AS851m Assunção, Wesley Klewerton Guez
ModelVars2SPL: An Automated Approach to Reengineer Model Variants into Software Product Lines / Wesley Klewerton Guez Assunção. – Curitiba, 2017.
185 f. : il. color. ; 30 cm.

Tese - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Ciência da Computação, 2017.

Orientadora: Silvia R. Vergilio - Coorientador: Roberto E. Lopez-Herrejon.

1. Ciência da computação. 2. Engenharia de Software. 3. Reengenharia. 4. Extração de LPS. I. Universidade Federal do Paraná. II. Vergilio, Silvia R. III. Lopez-Herrejon, Roberto E. IV. Título.

CDD: 005.133

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
Setor CIÊNCIAS EXATAS
Programa de Pós-Graduação INFORMÁTICA

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da tese de Doutorado de **WESLEY KLEWERTON GUEZ ASSUNCAO** intitulada: **ModelVars2SPL: An Automated Approach to Reengineer Model Variants into Software Product Lines**, após terem inquirido o aluno e realizado a avaliação do trabalho, são de parecer pela sua Aprovação.

Curitiba, 11 de Abril de 2017.

SILVIA REGINA VERGILIO
Presidente da Banca Examinadora (UFPR)

ROBERTO ERICK LOPEZ-HERREJON
Co-orientador - Avaliador Externo (ETS)

EDSON ALVES DE OLIVEIRA JUNIOR
Avaliador Externo (UEM)

MARTIN A. MUSICANTE
Avaliador Externo (UFRN)

THELMA ELITA COLANZI LOPES
Avaliador Externo (UEM)

MARCOS DIDONET DEL FABRO
Avaliador Interno (UFPR)

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
Setor de Ciências Exatas
Coordenação de Pós-Graduação em Informática

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor Prof. Silvia Regina Vergilio for the continuous support of my studies and research. I am sure that without your patience, motivation, enthusiasm, and immense knowledge, this journey wouldn't be possible. I could not have imagined having a better advisor.

My sincere thanks also goes to Prof. Roberto Erick Lopez-Herrejon for agreeing to supervise me during my studies in Linz, Austria. Your brought valuable contribution to my research. Thanks for being my co-advisor and contribute to my research career. I also would like to thanks Alexander Egyed for accepting to host me as a visiting student in the Institute for Software Systems Engineering (ISSE) at Johannes Kepler University (JKU).

Besides my advisors, I would like to thank the rest of the committee member: Prof. Martin Musicante, Prof. Edson Oliveria Júnior, Prof. Thelma Elita Colanzi Lopes, and Prof. Marcos Didonet Del Fabro, for their insightful comments and questions.

I thank my fellow labmates for the stimulating discussions, for the companionship in the conferences and for all the fun we have had in the last years. I would like to express my grateful for all staff of Federal University of Paraná for providing a fruitful environment to develop my research. Thanks also to the Brazilian Coordination for the Improvement of Higher Education (CAPES) for the scholarship and financial support.

Last but not the least, I would like to thank my family: *Obrigado pela educação e amor que me deram durante toda a minha vida e por terem me apoiado em todos os momentos*. Thanks to my wife Mileide: *Obrigado por me apoiar e incentivar durante todo o período dos meus estudos.*

Wesley Klewerton Guez Assunção
Curitiba-PR, Brazil. April 2017

Abstract

Software Product Lines (SPLs) are families of related software systems developed for specific market segments or domains. SPLs commonly emerge from sets of existing variants when their individual maintenance and/or evolution become complex. However, current approaches for SPL extraction from existing variants do not support design models, are partially automated, or do not reflect domain constraints in terms of feature combinations. To tackle these limitations, the goal of this work is to present an automated approach to reengineer model variants into an SPL, called ModelVars2SPL (**Model Var**iant**s to S**oftware **P**roduct **L**ine). The input of the approach is a set of *Unified Modeling Language* (UML) class diagrams and the list of features they implement. All the reengineering process is covered, and the output includes (i) a Feature Model, which represents the combinations of features of the input variants, and (ii) a Product Line Architecture, which represents a global architecture with feature-related annotations. The reengineering process of ModelVars2SPL is composed of four steps, two of them rely on search-based techniques and the others are based on deterministic algorithms. There is no need for human experts for obtaining solutions. We conducted an experiment to evaluate the quality of the solutions obtained with the proposed approach. The quality of the FMs and PLAs was measured by considering how well these artifacts represent the input variants. Furthermore, we evaluate the quality of the outputs in each step of the approach taking into account the goals of the reengineering process. For the experimentation we used ten case studies representing two different scenarios. The results of the evaluation show that the approach can obtain solutions with high degree of correctness in terms of representing the input variants, and that the outputs of the steps are in accordance to the phases of the reengineering process. Based on an example of use we show how the obtained FM and PLA make easier the maintenance activity.

Keywords: Reuse, Reengineering, Software Product Line, SPL extraction, Search-Based Software Engineering.

Resumo

Linhas de Produto de Software (LPSs) são famílias de sistemas de software relacionados que são desenvolvidos para um segmento de mercado ou domínio. Comumente, LPSs surgem de um conjunto de variantes existentes, quando a manutenção e/ou evolução individuais tornam-se complexas. Contudo, as abordagens encontradas na literatura para extração de LPS a partir de variantes existentes não dão suporte a modelos de projeto, são parcialmente automatizadas, ou não refletem restrições de domínio em termos de combinação de características. Para lidar com estas limitações, o objetivo deste trabalho é apresentar uma abordagem automatizada para fazer a reengenharia de variantes de modelos em uma LPS, chamada *ModelVars2SPL* (Variantes de Modelos para Linha de Produto de Software, do Inglês ***Model Var****iant****s to S****oftware **P****roduct **L****ine*). A entrada para a abordagem é um conjunto de diagramas de classe *Linguagem de Modelagem Unificada* (UML) e uma lista de características que estes implementam. Todo o processo de reengenharia é coberto, e a saída inclui (i) um Modelo de Características, que representa a combinação de características das variantes de entrada, e (ii) uma Arquitetura de Linha de Produto, que representa uma arquitetura global com características anotadas. O processo de reengenharia da *ModelVars2SPL* é composto por quatro passos, sendo dois deles apoiados em técnicas baseadas em busca, e os dois outros baseados em algoritmos determinísticos. Não existe a necessidade de especialistas humanos para obter soluções. Para avaliar a abordagem proposta, foi conduzido um experimento para aferir a qualidade das soluções obtidas. A qualidade dos Modelos de Características e das Arquiteturas de Linha de Produto foi medida considerando-se o quão bem as variantes de entrada foram representadas. Além disso, a qualidade das saídas em cada passo da abordagem foi avaliada levando-se em consideração os objetivos do processo de reengenharia. Para a experimentação utilizaram-se dez estudos de caso representando dois cenários diferentes. Os resultados da avaliação mostram que a abordagem consegue obter soluções com alto grau de corretude em termos de representação das variantes de entrada, e que as saídas dos passos estão de acordo com as fases do processo de reengenharia. Com base em um exemplo de uso de uma solução mostra-se como os artefatos de LPS obtidos facilitam a atividade de manutenção.

Palavras-chave: Reúso, Reengenharia, Linha de Produto de Software, Extração de LPS, Engenharia de Software Baseada em Busca.

Contents

1 Introduction **14**
- 1.1 Problem Statement . . . 16
- 1.2 Goal and Motivation . . . 16
- 1.3 Summary of Contributions . . . 17
- 1.4 Organization of the Dissertation . . . 18

2 Reengineering Existing Systems into SPLs **19**
- 2.1 Clone-and-Own . . . 19
- 2.2 Software Product Lines . . . 20
  - 2.2.1 Feature Models . . . 20
  - 2.2.2 Product Line Architecture . . . 22
- 2.3 Search-based Software Engineering . . . 23
- 2.4 Reengineering System Variants into SPLs . . . 23
  - 2.4.1 Reengineering Process . . . 23
  - 2.4.2 Related Work . . . 25
  - 2.4.3 Limitations of Existing Works . . . 25
- 2.5 Concluding Remarks . . . 27

3 The ModelVars2SPL Approach **28**
- 3.1 ModelVars2SPL Overview . . . 28
- 3.2 Features Traceability . . . 31
  - 3.2.1 Decomposition of Model Elements . . . 31
  - 3.2.2 Traceability Discovery . . . 32
  - 3.2.3 Dependency Graph . . . 34
- 3.3 Reverse Engineering of Feature Models . . . 36
  - 3.3.1 Representation . . . 36
  - 3.3.2 Fitness Functions . . . 36
  - 3.3.3 Evolutionary Operators . . . 38
  - 3.3.4 Output . . . 39
- 3.4 Model Merging . . . 41
  - 3.4.1 Representation . . . 41
  - 3.4.2 Fitness Function . . . 41
  - 3.4.3 Evolutionary Operators . . . 42
  - 3.4.4 Output . . . 43
- 3.5 Variability Grafting . . . 43
- 3.6 Guidelines for Using ModelVars2SPL Solutions . . . 44
- 3.7 Concluding Remarks . . . 46

4 Evaluation of the Proposed Approach **48**
4.1 Research Questions . . . 48
4.2 Measures to Evaluate the FM and the PLA . . . 49
4.3 Measures to Evaluate the Outputs of Each Step . . . 49
4.4 Case Studies . . . 50
4.5 Experimental Settings and Implementation Details . . . 54
4.6 Results and Analysis . . . 55
4.6.1 Features Traceability . . . 55
4.6.2 Reverse Engineering of Feature Models . . . 57
4.6.3 Model Merging . . . 60
4.6.4 Variability Grafting . . . 61
4.6.5 Discussion of the Results . . . 64
4.7 Example of Use of the ModelVars2SPL Solutions . . . 67
4.8 Threats to Validity . . . 68
4.9 Concluding Remarks . . . 69

5 Conclusion **70**
5.1 Contributions . . . 71
5.2 Research Limitations . . . 71
5.3 Future Research Directions . . . 72

Bibliography **74**

Appendix A Reengineering legacy applications into software product lines: a systematic mapping **83**

Appendix B Multi-objective reverse engineering of variability safe feature models based on code dependencies of system variants **129**

Appendix C Discovering Software Architectures with Search-based Merge of UML Model Variants **162**

Appendix D Case Studies Details **179**

List of Figures

2.1 SPLE framework, extracted from Apel et al. [7] 21
2.2 FMs graphical notation using FODA 22
2.3 The reengineering process 24

3.1 Proposed approach for reengineering systems to SPL 29
3.2 Example of input for the approach 30
3.3 Example of output of the approach 31
3.4 Example of overlap analysis 34
3.5 Example of dependency graph 36
3.6 FM metamodel, extracted from [69] 37
3.7 Example of reverse engineered FMs 40
3.8 Ecore metalmodel, extracted from EMF documentation 42
3.9 Example of merged UML class diagram 44
3.10 Example of variabilities in the PLA 45

4.1 Model elements of the variants in each case study 52
4.2 UML class diagram variants of BS 53
4.3 Dependency graph of BS 57
4.4 Composition of *Base* + *WithdrawLimit*, from BS1 to BS2 58
4.5 Solutions on the search space 59
4.6 FM and configurations of BS 60
4.7 Evolution of the best solution 62
4.8 Merged UML class diagram of BS 63
4.9 PLA of BS 64
4.10 Baselines and PLAs 66
4.11 Example of traceability between the FM and the PLA for BS 68

List of Tables

3.1 Atomic model elements of Variant2 . . . 33
3.2 Traceability of the illustrative example . . . 35
3.3 Feature sets of the illustrative example . . . 39
3.4 Dependency matrix of the illustrative example . . . 40

4.1 Case studies . . . 51
4.2 NSGA-II parameters . . . 54
4.3 gGA parameters . . . 55
4.4 Feature Traceability results . . . 56
4.5 Modules of BS . . . 57
4.6 Reverse Engineering of FMs results . . . 59
4.7 Model Merging results . . . 61
4.8 Variability Grafting results . . . 63
4.9 Results for the obtained FMs and PLAs . . . 65
4.10 Approach runtime . . . 67

D.1 Banking System (BS) . . . 180
D.2 Draw Product Line (DPL) . . . 180
D.3 Mobile Media - Version 1 (MMv1) . . . 180
D.4 Video On Demand (VOD) . . . 181
D.5 ZipMe (ZM) . . . 182
D.6 Game Of Life (GOL) . . . 183
D.7 Mobile Media - Version 2 (MMv2) . . . 183
D.8 Mobile Media - Version 3 (MMv3) . . . 184
D.9 Mobile Media - Version 4 (MMv4) . . . 184
D.10 Mobile Media - Version 5 (MMv5) . . . 185

Publications

A list of publications is presented next. This list includes journal articles, conference full papers, and a short paper of a doctoral symposium. All these publications are results of studies conducted during the doctoral term. Some of these studies are directly related to this dissertation, they are properly referenced in the text and included as appendixes. Other studies are indirectly related, since all of them involve Search-Based Software Engineering, Software Product Lines, or UML models/diagrams.

Journals

1. ASSUNÇÃO, W.K.G., Lopez-Herrejon, R.E., Linsbauer, L., Vergilio, S.R., Egyed, A. Reengineering Legacy Applications into Software Product Lines: A Systematic Mapping. Empirical Software Engineering (2017). doi:10.1007/s10664-017-9499-z. (*Appendix A*)

2. ASSUNÇÃO, W.K.G., Lopez-Herrejon, R.E., Linsbauer, L., Vergilio, S.R., Egyed, A. Multi-objective reverse engineering of variability-safe feature models based on code dependencies of system variants. Empirical Software Engineering (2016). doi:10.1007/s10664-016-9462-4. (*Appendix B*)

3. ASSUNÇÃO, W.K.G.; Colanzi, T.E.; Vergilio, S.R.; Pozo, A.T.R. A multi-objective optimization approach for the integration and test order problem. Information Sciences, v. 271, p. 1-21, 2014.

4. ASSUNÇÃO, W.K.G.; Barros, M. de ; Colanzi, T.E. ; Dias-Neto, A.C.; Paixão, M.H.E.; De Souza, J.T. ; Vergilio, S.R. A mapping study of the Brazilian SBSE community. Journal of Software Engineering Research and Development, v. 2, p. 3, 2014.

5. ASSUNÇÃO, W.K.G.; Colanzi, T.E.; Vergilio, S.R.; Pozo, A.T.R. Evaluating different strategies for integration testing of aspect-oriented programs. Journal of The Brazilian Computer Society, v. 20, p. 9, 2014.

6. ASSUNÇÃO, W.K.G.; Colanzi, T.E.; Vergilio, S.R.; Pozo, A.T.R. Generating Integration Test Orders for Aspect Oriented Software with Multi-objective Algorithms. Revista de Informática Teórica e Aplicada: RITA, v. 20, p. 301-327, 2013.

7. Colanzi, T.E.; ASSUNÇÃO, W.K.G.; DE Freitas G.T.D.; Zorzo, C.A.; Vergilio, S.R. Evaluating Different Strategies for Testing Software Product Lines. Journal of Electronic Testing. Dordrecht, v. 29, p. 9-24, 2013.

Conferences - Full Papers

1. ASSUNÇÃO, W.K.G.; Lopez-Herrejon, R.E.; Vergilio, S.R. Discovering Software Architectures with Search-based Merge of UML Model Variants. In: 16th International Conference on Software Reuse (ICSR), 2017, Salvador, Brazil. To appear. (*Appendix C*)

2. ASSUNÇÃO, W.K.G.; Lopez-Herrejon, R.E.; Linsbauer, L.; Vergilio, S.R.; Egyed, A. Extracting Variability-Safe Feature Models from Source Code Dependencies in System Variants. In: Genetic and Evolutionary Computation Conference (GECCO), 2015, Madrid, Spain. (*Best Paper Award nomination*)

3. Lopez-Herrejon, R.E. ; Linsbauer, L. ; ASSUNÇÃO, W.K.G.; Fischer, S. ; Vergilio, S.R. ; Egyed, A. Genetic Improvement for Software Product Lines: An Overview and a Roadmap. In: First International Genetic Improvement Workshop (GI) - Genetic and Evolutionary Computation Conference (GECCO), 2015, Madrid, Spain.

4. ASSUNÇÃO, W.K.G. ; Vergilio, S.R. Feature Location for Software Product Line Migration: A Mapping Study. In: 2nd International workshop on Reverse Variability Engineering (REVE) - 18th International Software Product Line Conference (SPLC), 2014, Florence, Italy.

5. ASSUNÇÃO, W.K.G. ; Vergilio, S.R. Class Diagram Retrieval with Particle Swarm Optimization. In: 25th International Conference on Software Engineering and Knowledge Engineering (SEKE), 2013, Boston, USA.

6. ASSUNÇÃO, W.K.G. ; Colanzi, T.E. ; Vergilio, S.R. ; Pozo, A.T.R. On the Application of the Multi-Evolutionary and Coupling-Based Approach with Different Aspect-Class Integration Testing Strategies. In: Symposium on Search-Based Software Engineering (SSBSE), 2013, St. Petersburg, Russia.

7. ASSUNÇÃO, W.K.G.; Vergilio, S.R. A Multi-objective Solution for Retrieving Class Diagrams. In: 2nd Brazilian Conference on Intelligent Systems (BRACIS), 2013, Fortaleza, Brazil.

8. ASSUNÇÃO, W.K.G.; Colanzi, T.E.; Vergilio, S.R.; Pozo, A.T.R. Determining Integration and Test Orders in the Presence of Modularization Restrictions. In: 27th Brazilian Symposium on Software Engineering (SBES), 2013, Brasília, Brazil.

9. ASSUNÇÃO, W.K.G.; Barros, M.; Colanzi, T.E.; Dias-Neto, A.; Paixão, M.; Souza, J.; Vergilio, S.R. Mapeamento da Comunidade Brasileira de SBSE. In: IV Workshop de Engenharia de Software Baseada em Busca (WESB) - Congresso Brasileiro de Software: Teoria e Prática (CBSoft), 2013, Brasília, Brazil.

Doctoral Simposium

1. ASSUNÇÃO, W.K.G. Search-Based Migration of Model Variants to Software Product Line Architectures. In: 37th International Conference on Software Engineering (ICSE) - Doctoral Symposium, 2015, Florence, Italy.

Chapter 1

Introduction

Developing software systems from scratch is a complex and high cost activity. In industry, programmers, engineers, and architects adopt strategies to reduce the cost and improve the quality in the development of new software systems. A well established strategy is software reuse, which is based on the use of existing artifacts to develop new software systems [59]. Among the advantages provided by software reuse we can mention the reduction of time-to-market, an improved productivity, and increased quality [31]. Any artifact built during the software development can be reused, including source code, design models, test cases, etc.

In practice, software reuse is generally performed using an ad hoc strategy, called *clone-and-own* (also known as copy-paste-modify) [87]. Clone-and-own approach was identified by a study as the most common reuse scenario in practice [38]. In this approach, when there is a demand for new functionalities in a system, existing software artifacts are cloned and adapted to fulfill the new requirements. The clone-and-own approach is an easy way to reuse software and does not require an upfront investment, while obtains good short-term results quickly.

Clone-and-own helps the development of new system variants, however, the main disadvantage of this approach is the simultaneous maintenance and evolution of a typically large number of variants [43]. In this scenario, as the number of variants and functionalities increases so does the complexity of their development and maintenance, hence a systematic reuse approach is necessary. A way to tackle this problem is the adoption of a Software Product Line approach [85].

A *Software Product Line (SPL)* is a set of software products that share common features, and are designed for a specific domain [25, 66]. A *feature* is a user-visible aspect, functionality, or characteristic of a system [56]. The main advantage of an SPL is the systematic reuse of a common infrastructure, which is shared to create different product variants. *Software Product Line Engineering (SPLE)* is the discipline of developing and managing SPLs. SPLE identifies two types of assets: the *common assets* (also known as commonalities) reused in all products, and the *variable assets* (also known as variabilities) related to those features that are provided only by some products. In addition, SPLE deals with the effective management of SPLs throughout their entire life cycle.

According to Krueger [60], there are three ways to undertake SPLE: (i) from scratch, by applying a complete domain analysis and variability management in advance of any application engineering, called *proactive* approach; (ii) by creating and updating the SPL when every new product appears, the *reactive* approach; and (iii) by using an *extractive* approach, which takes existing products to extract common and variable assets.

In this context, the extractive approach is the most common way to systematize the software reuse with SPLs [63]. The extractive approach encompasses the reengineering of

existing systems, leading to a systematic reuse and easier maintenance and evolution, because the systems are not maintained or evolved individually but as a group considering both commonalities and variabilities. Besides these technical benefits, the reengineering of existing systems into an SPL allows companies to preserve their investment and aggregate knowledge obtained during the development of the variants. Because of these reasons, the extractive approach has attracted interest from companies as well as researchers [72].

To cover the state-of-art on the field of reengineering existing systems into SPLs we conducted a systematic mapping study [10], which is presented in Appendix A. In this study we identified that the reengineering process of the extractive strategy of the SPLE is composed of three phases:

1. *Detection phase*, where the relevant information between features and implementation artifacts is gathered. The goal is to identify where the features are implemented. Traceability links are the most common information obtained in this phase [68].

2. *Analysis phase*, devoted to the identification and organization of the similarities and variabilities in a variability model. A *Feature Model (FM)* is the most widely used variability model to describe the possible combinations of features in an SPL [56].

3. *Transformation phase*, responsible for realizing the common and variable parts of an SPL in the corresponding assets. An artifact to be achieved here is the Product Line Architecture. A *Product Line Architecture (PLA)* enables the variability abstraction in the design process describing what is common and variable to all the products of the SPL [30].

The reengineering process is a complex activity because domain constraints (i.e., feature constraints) are not always explicit. Furthermore, the number of variants and features can difficult the process when done without the support of specialized tools. These reasons lead researchers to deal with some phases of the reengineering process by leveraging existing strategies, such as feature location, execution tracing, information retrieval, model transformations, aspect oriented programming, search-based techniques, etc. More details about these strategies can be found in our mapping study [10].

The benefits of employing a systematic reuse to manage and evolve a set of system variants are well known, however, in the literature we can find only a limited number of approaches to deal with the reengineering process [10]. For instance, an approach called PuLSETM-DSSA integrates different software artifacts to obtain a reference architecture, which guides the migrating of system variants into an SPL [16]. An algorithm named ThreeVaMaR is used to refactor Unified Modeling Language (UML) models identifying common parts and the variable ones to generate a reference model and to support the reengineering process [89]. A method proposed by Nie et al. creates a variability model for each system variant and then merges all the variability models in a unique artifact [79]. In the approach proposed by Kumaki et al., an SPL expert analyzes similarities and differences in a set of model variants and in a set of requirements to generate a variability model and a reference architecture [62]. A semi-automated approach called MoVa2PL identifies common and variable blocks of elements from a set of model variants, and then these parts are used to create a model-based SPL [73].

Despite the existence of some works to deal with the reengineering process, in our mapping study we observed some limitations of the existing approaches [10]. These limitations are described in the following section.

1.1 Problem Statement

The simultaneous maintenance and evolution of a typically large number of individual system variants developed by the using clone-and-own approach is a complex and error-prone activity. This happens because each variant has both distinct and duplicated implemented functionalities [43]. Furthermore, frequently programmers, engineers, and architects do not have a global architecture that shows how the different implementations are spread along the variants (i.e., there is a lack of a global view). In these cases of lacking support for ad hoc reuse, a systematic reuse approach is paramount, however the challenge is to find an approach that best fits the industrial scenario.

Thanks to the analysis of the works we found in our mapping study, we identified a wide number of research opportunities and gaps still opened in the research topic of reengineering existing systems into SPLs [10]. The following limitations were observed in the literature:

1. *Limitations of automation and tool support*. Few tools are available to support the reengineering process. Most approaches require extensive human expertise.

2. *Limitations on the used artifacts*. Most related works focus on source code artifacts. We could observe that most of them deal with the artifacts in a low level of abstraction. There is a lack of approaches to consider other sources of information (e.g., UML design models) as input for the reengineering process.

3. *Limitations related to the reengineering process*. Most studies focus on specific parts of the reengineering process, but in practice it must be considered in its entirety. For instance, some works have focus only in the traceability between features and implementation artifacts.

4. *Limitations related to feature management*. Feature management is challenging because of the large number of possible feature combinations in an SPL. Current approaches generally do not consider domain constraints among features. This fact hinders the consistent instantiation of products.

Based on the limitations presented above, our focus on this doctoral work is to deal with the problem of automatically obtaining an FM and a PLA from a set of model designs, covering the entire reengineering process, and taking into account domain constraints existing in the variants under consideration.

The solution for this problem is not simple or trivial. An automated approach for the entire reengineering process requires some fundamental phases that have different goals and deal with specific information and artifacts. Sometimes the input artifacts have implicit information or are partially incomplete, for example, in some cases the domain constraints existing in terms of feature combinations are not clearly described, or only a small set of variants are available. These situations require a well designed strategy. Furthermore, design models are complex and many times are very large artifacts to describe system structures. Dealing with many of these artifacts, each one representing a different variant, may be challenging.

1.2 Goal and Motivation

The general goal of this doctoral work is to propose and implement a fully automated approach to reengineer a set of design models variants into an SPL, in terms of FMs and PLAs

(*Limitations 1 and 2*). This approach covers the entire reengineering process (*Limitation 3*), producing intermediate artifacts in each step of the process. In addition, our approach considers existing domain constraints in combinations of features (*Limitation 4*).

The objective on implementing an automated approach is to motivate and support the adoption of systematic reuse in practice. Given the fact that most approaches are expert-dependent, this can hamper their adoption in the industry because of the required additional effort and investment.

Regarding design models, we deal with UML models, motivated by their widely use in industry and academia. The UML is the de-facto standard formalism for the design and analysis of software systems. More specifically, our approach uses as input a set of UML class diagrams, one of the most important components of UML [20]. Class diagrams are used to represent software architectures describing structural aspects of software systems [55]. In the context of this dissertation, a PLA is a design model defining the static architecture of the family of systems through a maximal UML class diagram and feature-related annotations.

Using as input UML class diagram variants we can support the whole reengineering process. Independently of the programming language, software engineers can use the design of the system to perform the reengineering. Furthermore, the output of our approach is an SPL architecture. Software architectures are artifacts that provide a high-level view of functional parts of systems and allow analyzing their structure [37]. Another objective is to integrate our work in the context of Model-Driven Software Development of SPLs [104].

To incentive the adoption of systematic reuse in practice, our approach covers the entire reengineering process; namely detection, analysis, and transformation. The process starts from artifact variants and extracts FMs and PLAs. But instead of generating only SPL artifacts, our approach generates outputs for each phase of the reengineering process, allowing programmers, engineers, and architects to follow and understand the process.

The objective for the feature management is to preserve the characteristics of existing product variants in the generated SPL artifacts. Feature constraints, which are implicit in the variants are, represented in the SPL artifacts. Having the FM and PLA of existing products, programmers, engineers, and architects can decide how to proceed with the maintenance and evolution of existing variants.

1.3 Summary of Contributions

The first contribution of this doctoral work is the mapping of the field of reengineering existing variants into SPLs [10, 12]. The mapping study was conducted as an initial step of this doctoral work to identify works and research opportunities. As a result of the mapping study, we proposed a definition for the reengineering process, we described the main strategies used in the reengineering process, the common inputs and outputs used, existing tools, and presented a wide set of research opportunities.

Regarding to the proposed approach, our contributions are present in the entire reengineering process. In the *detection phase*, we perform the traceability between features and implementation artifacts based on UML class diagrams, extending an existing tool that initially supported mainly source code. In the *analysis phase*, our study on reverse engineering of FMs introduces the multi-objective perspective of the problem and considers dependencies in the source code [8, 9]. In addition to the contribution of the previous work considering source code, in this dissertation we deal with the problem of reverse engineering of FMs considering dependencies existing between elements of UML models. With respect to the *transformation phase*, we present a new method to merge UML models using a search-based technique [11].

Furthermore, in this doctoral work we introduce the task of variability grafting in a merged UML model to leverage the adoption of SPLs.

The last contribution refers to the evaluation of the proposed approach. In the experimental evaluation of the approach we present evidence with respect to the reengineering of variants into SPLs at the design level. Furthermore, we provide guidelines regarding the practical use of the solutions.

1.4 Organization of the Dissertation

This dissertation is organized as follows: Chapter 2 contains a background covering relevant concepts used along the text and related work. Chapter 3 is devoted to the details of the approach. The evaluation setup of the proposed approach, the results and analysis are presented in Chapter 4. Chapter 5 presents concluding remarks, contributions, and future directions to new research. Appendices A to C present the studies produced as result of this doctoral work and that are relevant to this dissertation, and Appendix D has details of case studies used in the evaluation.

Chapter 2

Reengineering Existing Systems into SPLs

This chapter reviews terminology and main concepts used throughout the dissertation. First we introduce the Clone-and-Own approach and describe the problems related with ad hoc practices for reuse. The SPL approach is presented in the second section. Next, we introduce concepts of the Search-Based Software Engineering area. Then, we present the results of our systematic mapping study, which encompasses the definition of the reengineering process, the related work, and limitations of existing approaches. The last section presents concluding remarks.

2.1 Clone-and-Own

The most common scenario of software reuse in practice is the application of ad hoc techniques. For instance, when there exists demand for a new product with similar functionalities to an existing one, usually developers copy the new product from other already existing software and then adapt it to fit the new requirements. These ad hoc practices are known as *Clone-and-Own* (C&O) [91].

C&O is a simple and efficient way to reuse software artifacts, but products developed following ad hoc practices have their own and independent development life cycle. C&O approaches may work fine with small number of products, depending on products complexity, the development organization and its software engineering practices. However, in most situations, adding new products is no longer feasible either because of managerial, economical, or technical reasons. For example, the maintenance of many independent products leads to multiple problems such as inefficient feature update or bug fixing, duplicated functionality, and redundant/inadequate testing [38].

Besides the problems regarding maintenance of duplicated software artifacts, when we have to deal with a set of system variants, the problem of variability management raises. *Variability* is the capacity of software artifacts to assume different behaviors, and variability management is the ability to deal with an increasing amount of variant artifacts without losing their reuse capacity [22]. Variability management in a scenario with multiple system variants faces several issues, for example extracting variability from technical artifacts, tool support, design decisions management, testing of artifacts with variability, or domain design [23, 77].

The problems that emerge with the use of C&O must be solved to allow companies keeping many software products in their portfolio. The SPL approach is the premier alternative to cope effectively with these problems. This approach is presented in the next section.

2.2 Software Product Lines

Software Product Line (SPL) is "*a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission*" [25]. Over the last two decades extensive research and practice have been done in the field of SPLs [54]. The benefits provided by SPL practices are better customization, improved software reuse, and faster time to market. The basis of the approach is that the products are built using a core asset base instead of being developed one by one from scratch [54]. Members of an SPL are distinguished by the set of features they provide [85]. A *feature* is "*a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems*" [56]. Features are the building blocks of SPLs.

Software Product Line Engineering (SPLE) is the discipline responsible to develop SPLs. SPLE exploits the *commonality* (i.e., a property shared by all products of an SPL) and the *variability* (i.e., the capacity of different products of the SPL to vary). An effective management and realization of variability is at the core of successful SPL development [100]. Apel et al. proposed a feature-oriented process to undertake SPLE, this process is illustrated in Figure 2.1 [7]. According to this process, the development of SPLs has two main activities:

- The *domain engineering* (at the top of Figure 2.1), is composed of domain analysis, which includes tasks of domain scoping and variability modeling; and domain implementation, which means the representation of commonalities and variabilities in the implementation artifacts. The goal of this activity is to represent the domain using SPL assets;

- The *application engineering* (at the bottom of Figure 2.1), is composed of requirements analysis, responsible for combining features according to customers requirements; and product derivation, which includes the derivation of specific products and their corresponding validation and verification. The goal of this activity is to build products by reusing domain artifacts and exploiting the variability defined in the previous process (i.e., using SPL assets).

Krueger reported three ways used by companies as starting point to SPLE [60]:

- The *proactive approach*: first engineers perform a complete domain engineering, to have a full scope of products, then they develop reusable domain artifacts, and finally they use these artifacts for application engineering;

- The *reactive approach*: engineers incrementally grow their family of products by applying both domain and application engineering every time a new product is developed;

- The *extractive approach*: engineers use existing custom software system variants by extracting the common and varying artifacts, to migrate them to an SPL.

The extractive approach is the most common way to adopt SPLs in companies with many software system variants in production [60].

2.2.1 Feature Models

Feature Models (FMs) are a de facto standard for modeling the different combinations of features desired in an SPL and describing the variabilities [56]. The FM helps the communication among programmers, engineers, and architects since it provides a high level view of feature

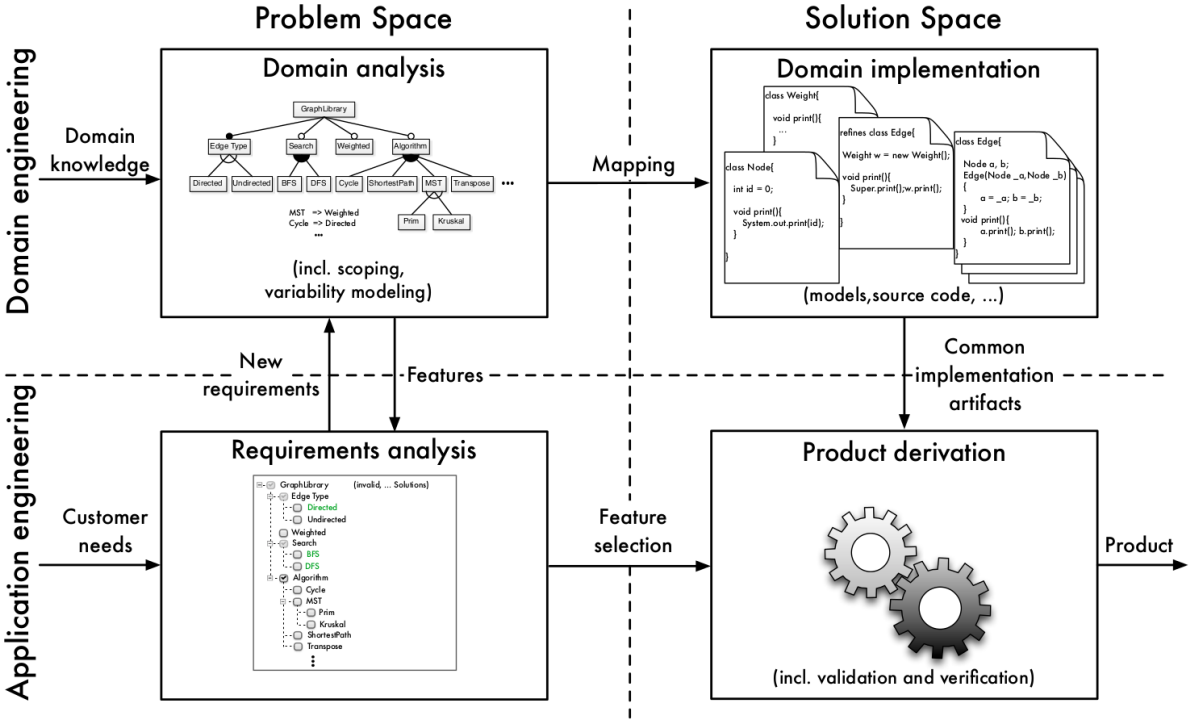


Figure 2.1: SPLE framework, extracted from Apel et al. [7]

combinations. In the FODA FM diagram notation, features are depicted as labeled boxes connected by lines to other features they relate with, all collectively forming a tree-like structure.

A feature can either be *mandatory*, which is selected in a system whenever its parent feature is also selected, or *optional*, which may or may not be selected whenever its parent feature is selected. They are respectively represented with filled and empty circles at the end of the feature relation as shown in Figures 2.2(a) and 2.2(b).

Features can be grouped into *alternative groups* or into *or groups*. In an *alternative group* if the parent feature of the group is selected then *exactly* one feature from the group must be selected. In an *or group* if the parent feature of the group is selected then *one or more* features from the group can be selected. Feature groups are depicted with lines connecting the parent feature (`P`) with the group features (`C1`, `C2`, and `C3`), crossed by an empty arc for *alternative* groups and by a filled arch for *or* groups as illustrated in Figures 2.2(c) and 2.2(d), respectively.

Besides the hierarchical relations among features, features can also relate across different branches of the FM with *Cross-Tree Constraints (CTCs)* [18]. The most common types are *requires/implies* relation whereby if a feature `A` is selected a feature `B` must also be selected, and *excludes* relation whereby if a feature `A` is selected then feature `B` *must not* be selected, and vice versa. These relations are commonly described by using propositional formula. Figure 2.2(e) illustrate two possible ways to describe the requires/implies relation, and Figure 2.2(f) presents two ways to describe the excludes relation.

FMs have an important role in the SPL development. When the FM reflects correctly the requirements and the needs of the stakeholders, then the software structure may be easily derived. Otherwise, the software structure should be further refined from the other domain products [56].

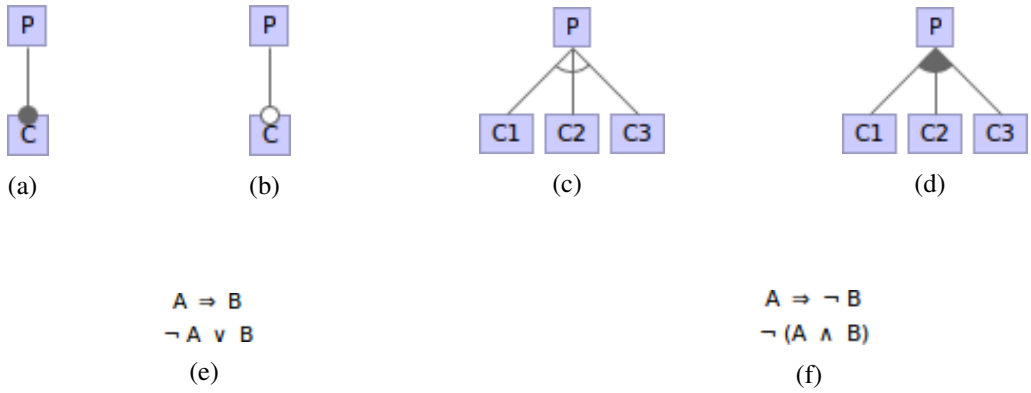


Figure 2.2: FMs graphical notation using FODA

2.2.2 Product Line Architecture

As mentioned before, the SPL approach is a premier alternative to solve problems related to maintenance and evolution of a set of system variants. However, to achieve these results, an SPL must be well represented allowing a global view and analysis of its products. Usually, software systems are represented using software architectures, which are important in software development because they work as a bridge between the requirements and the implementation. Several works mention about reference architectures as a way to represent a variety of different systems [26, 75]. Despite of having the purpose of representing software elements that support development of systems for a specific domain, a reference architecture has a broader goal that encompasses knowledge regarding business rules, architectural styles, or best practices of software development [78].

In the context of SPLs, there is a special type of architecture, called Product Line Architecture, which explicitly describes commonality and variability and is the basis for the architectures of all SPL products. *Product Line Architectures (PLAs)* enable the variability management in the design process and provide a high level of abstraction for its understanding. This architecture describes all the mandatory and varying implementation elements of its SPL domain [30, 101]. In addition to software architectures of monolithic systems, PLA is responsible for describing the scope of a well defined set of related products.

According to Bartory, "*software complexity is growing at an alarming rate and the costs of software development and maintenance must be restrained. PLAs enable companies to amortize the effort of software design and development over multiple products, thereby substantially reducing costs*" [15].

Although the importance of the PLA be clearly pointed, there is no standard representation for this architecture. With respect to our work, a PLA is a design model defining the static architecture of the family of systems through a maximal UML class diagram and feature-related annotations.

Employing the extractive approach of SPLE, SPL artifacts such as PLAs and FMs can be obtained using reengineering techniques. The extraction of SPL from existing artifacts is performed by a reengineering process, which is subject of Section 2.4. Some phases of this process can use search-based techniques from the field known as Search-based Software Engineering, which is presented in the next section.

2.3 Search-based Software Engineering

Search-based Software Engineering (SBSE) is a research area that addresses software engineering problems by using search-based optimization algorithms [49]. The motivation for SBSE is the fact that many software engineering problems have multiple conflicting and competing objectives, which must be considered simultaneously. Furthermore, search-based techniques (i.e., optimization algorithms) are known to perform well in cases with partial, noisy and missing data. These characteristics of the SBSE approaches entice the software engineering community to solve complex problems or to deal with noisy and incomplete data [51].

SBSE techniques have been widely adopted throughout the software engineering life-cycle [29]. Its applications range from software requirements selection to automated software repair [52]. Taking into account the scope of this dissertation, search-based techniques have been successfully used in tasks of reverse engineering with focus on migration [50]. The goal of applying optimization algorithms in reverse engineering is to obtain new software artifacts from existing ones. For instance, the discovery of software architectures from the source code implementation [86].

A survey on SBSE for SPLs was reported by Harman et al. [48]. This study presents many applications of search-based techniques in activities of SPLE. The growing interest on the use of SBSE techniques in SPLE is clearly pointed by the authors. According to this survey, the most pieces of work address SPL testing. Several studies deal with feature selection and FM construction. The improvement of internal quality of PLAs was the focus of some studies. Another activity of SPLE already explored by SBSE researchers is the feature extraction from software artifacts.

We presented a roadmap of the application of Genetic Improvement, a search-based technique, in the context of SPL [70]. We argue that there is a wide field in SPLE research to benefit from SBSE. For instance, a better sensitivity analysis of similarity among system variants, inclusion of UML models as a source of information for the identification of variabilities, and further exploration of repair mechanisms and software transplantation. For further details refer to our analysis of this research field [70].

Based on the aforementioned reasons, we can observe the advantages of using SBSE techniques in complex tasks of SE. However, a well designed SBSE solution is required to achieve the desired results. Three ingredients are necessary to implement a SBSE solution [52]: (i) an appropriate way to represent solutions, (ii) a function designed to evaluate the quality of a solution, and (iii) a set of operators to generate new solutions and explore the search space. In next chapter we make use of these ingredients in different steps of our proposed approach.

2.4 Reengineering System Variants into SPLs

As part of this doctoral work, we conducted a systematic mapping study to identify works in the field of reengineering system variants into SPLs and cover the state-of-art on this research topic [10]. The publication with the details of the mapping process and the complete results can be found in Appendix A. Next we describe the main results with respect to the scope of this dissertation.

2.4.1 Reengineering Process

According to Chikofsky and Cross, *reengineering* is "*the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the*

new form" [24]. Sometimes this term is confused with the term reverse engineering. However, the same authors define *reverse engineering* as "*the process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction.*" We can see that reverse engineering is concerned with understanding the systems, on the other hand, reengineering is concerned with restructuring/refactoring the systems. In this sense, reverse engineering is a prerequisite to the reengineering process, since we need to understand the subject system that we have to transform [35].

Until the publication of our mapping study there was not an established or widely accepted set of phases to conduct the reengineering process. As a result of our mapping, we proposed a process to reengineer systems variants into SPLs. An overview of the reengineering process with focus on SPLs is illustrated in Figure 2.3. Depicted on the left side of the figure we have the systems developed following C&O practices. The solid line represents the entire process of reengineering, which is usually composed of different phases, presented with dashed lines in the illustrative figure.

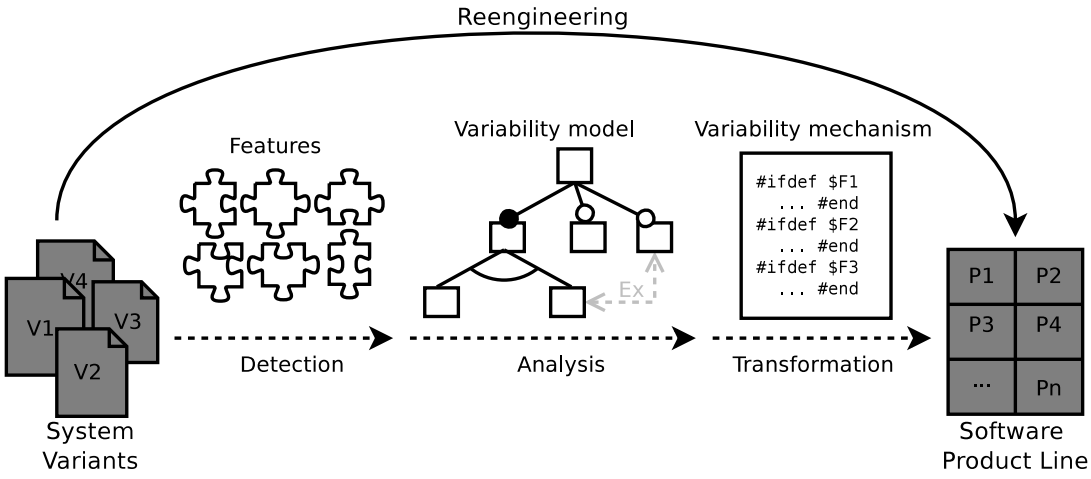


Figure 2.3: The reengineering process

The generic phases of the reengineering process of existing systems into SPLs are: (i) *Detection phase* is the beginning of the process, devoted to detect the variabilities and commonalities among existing products. The variabilities and commonalities are represented in terms of features as shown in Figure 2.3. Common support in this phase is given by *feature location* techniques, which aim at locating the elements responsible for implementing the system functionalities. The management of the features and the mapping/traceability to the software artifacts that implement them are tasks of the SPLE domain engineering process [36, 90] (see Section 2.2); (ii) *Analysis phase* involves the organization of discovered variabilities and commonalities. This step is devoted to create the variability model, shown in the middle of Figure 2.3, to express the valid combinations of features of an SPL. FMs are the most popular form of variability model (see Section 2.2.1); and (iii) in the *Transformation phase* the artifacts that implement the features and the variability model are used to create the SPL, using a variability mechanism. For instance, the simplest mechanism is based on `#ifdef` source code annotations made to the artifacts that are pre-processed, shown on the right side of Figure 2.3, following the desired feature selection, to create different products [13]. Another way to describe the variable and common parts of an SPL is by using a PLA (see Section 2.2.2).

The reengineering process may become a complex activity when the input variants do not describe well their commonalities and variabilities. For instance, sometimes there is no

explicit description regarding feature combinations constraints, or the implementation artifacts are poorly described in terms of their corresponding features. To solve problems like this, approaches from the SBSE area have been widely used [52].

2.4.2 Related Work

In this section we describe the pieces of work related to this doctoral work and main differences from our approach. From the works found in our mapping study [10], here we describe only those that have the same focus of our work, namely those which cover the entire reengineering process and deal with design models.

Bayer et al. introduce an approach, called PuLSETM-DSSA, to integrate software artifacts in different abstraction levels to obtain a reference architecture [16]. The reference architecture gives guidance for migrating the system variants into an SPL. Differently from our approach, PuLSETM-DSSA is not fully automated, requiring human expertise. Furthermore, PuLSETM-DSSA considers a wider range of artifacts as input, such as business goals and requirements, source code, and design models.

Rubin and Chechik propose a method based on refactoring of structural and behavioral UML models to merge variants into SPLs [89]. Their algorithm, named ThreeVaMaR, takes as input a model with duplications, which represent multiple product variants. ThreeVaMaR applies refactoring operations in the input model in order to put together common parts and to make as optional the variable part. Our approach also performs model merging to obtain a reference model, but in addition, we also generate an FM to describe the relationship among features.

Nie et al. present a method of model merging based on FMs [79]. The goal of their approach is to create an FM for each system variant and then merge all FMs to obtain the variability model. The resulting variability model denotes the feature combinations of all system variants. This approach does not take into account the merging of UML models or the construction of PLAs, which is the goal of our approach.

The work of Kumaki et al. has a goal similar to our work, which is to generate a variability model and a architecture [62]. The input of their approach is also a set of UML class diagram variants, but instead of using feature sets, they use a set of requirements. To identify common and variable sentences in the requirements their approach uses overlap analysis. The same overlap analysis is done in the class diagram variants. An algorithm based on vector space model is applied to recommend traceability between requirement sentences and class diagram elements. Using this traceability, an SPL expert develops the variability model and a reference architecture that best represents the variants. Despite that the approach has some points similar to our approach, it requires human effort, differently from our approach, which is fully automated.

Martinez et al. propose an approach called MoVa2PL to generate model-based SPLs [73]. MoVa2PL first identifies common and variable blocks of model elements from a set of model variants, then these blocks of model elements are mapped manually to their corresponding features. The model-based SPL is obtained by describing variabilities using the Common Variability Language [21]. Our approach differs from MoVa2PL because the traceability between model elements and features is done automatically.

2.4.3 Limitations of Existing Works

In the previous section we presented related work and the main differences between them and the approach being proposed in this doctoral work. However, in our systematic mapping we have also identified a set of limitations and research opportunities in the field of reengineering

systems variants into SPL [10]. Taking into account the goal of this dissertation, next we describe in details some limitations we addressed with the approach proposed.

Automated Support to the Reengineering Process

We can observe in the literature a lack of approaches to automate the entire process of reengineering of existing variants into SPLs (Limitations 1 and 3 presented in the introduction, Section 1.1). Most approaches are semi-automatic, and highly human dependent. Many studies expose only an intention to provide an automated support to their methods. The first reason to provide an automated support to the reengineering process is to reduce the manual effort [1, 76, 99, 109]. An automated process can improve the overall quality of the reengineering process, since this process is a labor-intensive task and error-prone [99]. In this sense, authors argue for the necessity of providing fully automated support, such as [2, 33, 82, 92, 111], enabling an easier and better application of their approaches.

Despite the need to automate the entire reengineering process, some authors point out the missing tool support for specific tasks. For instance, for the detection phase, Santos et al. suggest as future research the use of test-based feature location to automate the mapping of features to source code [93]. For the analysis phase, Xue et al. [108] mention the possible use of tools to automate the reconciliation of inconsistent FMs and Li et al. [64] argue that there is still no tool for feature aggregation and abstraction. Regarding the transformation phase, Olszak and Jørgensen point out the labor-intensive task of manually annotating feature entry points [80]. She et al. describe as challenge the automation of feature location and dependency mining when the focus of reengineering is large-scale systems [96].

In summary, given the increasing interest in SPLs, the implementation of automated approaches to support the entire reengineering process is fundamental to the practice and use in the industry. The approach proposed in this dissertation is designed to provide an automated support to all phases of the reengineering process.

Exploiting Multiple Sources of Information for Reengineering

Another research direction observed is exploiting different information sources during the reengineering process (Limitation 2 presented in Section 1.1). A research opportunity presented by Knodel et al. is the use of test cases, commonly available in most projects, in conjunction with other sources to determine features [58]. Trifu argues for the extraction of direct flow relations from sources other than the source code [103]. Kelly et al. suggest exploring source code comments and documentation to enrich their approach for concept mining [57]. Eyal-Salman et al. indicate as future work the use of relationships between source code elements to improve the traceability and feature identification [41]. Duszynski et al. [40] and Peng et al. [83] mentioned as research direction the use of design knowledge such as architecture models to allow the reengineering at a higher abstraction level. Bécan et al. do not point out what specific source should be explored, but recommend that all artifacts that may be present in software projects can be used [17]. In the same way, Yu et al. envisage the use of multi-grained resources, such as code bases, historical code changes, mailing lists, bug databases, software descriptions, user evaluations, etc. [110]. To have a benefit in using different sources of information, Kulesza et al. propose that links between the different artifacts should be constantly managed [61]. This enables the use of different sources in conjunction, such as proposed by Almeida et al. [6], who recommend as future work the use of both domain analysis and domain design in the software evolution. In the same way She [95] suggests the combination of bottom-up vs. top-down synthesis using artifacts at different levels of abstraction to cover different viewpoints.

Our proposed approach exploits the use of design models, namely UML class diagrams, as a source of information to the reengineering process. Using as input UML class diagram variants we can support the reengineering process independently of the programming language. Furthermore, using design models the programmers, engineers, and architects can have a broader view of the SPL.

Feature Management

Feature management is an important task in the reengineering process, responsible for providing the variability among the features that compose the product variants. However, in the literature there are still open gaps (Limitation 4 presented in Section 1.1).

An identified trend is the automated recovery of feature dependencies and interactions considering aspects such as non-functional characteristic of systems [5, 14, 64]. These aspects may help to refine the feature mappings and improve the resulting SPL [65]. Many studies generate FMs as output but, in general, constraints, such as one feature requires or excludes another feature, are not considered [3, 4, 32, 42, 44, 53, 97, 107, 112]. Automatic recovery of constraints is an open issue to be addressed in new studies. Another research direction is the reengineering of partial variability, where a subset of features with variability are considered more important and hence should be given priority in the reengineering process (e.g., they will be migrated first, ahead of other lower priority features) [71, 88].

Among the open gaps in feature management, the approach proposed in this dissertation is designed to deal with constraints existing in combination of features. Our approach identifies features that always appear together and features that exclude each other. This enables the approach to represent better the input variants in an SPL.

2.5 Concluding Remarks

The use of C&O approaches to reuse software artifacts leads to problems related to maintenance and evolution of system variants developed independently. In this context, the SPL approach is a premier alternative to systematize the software reuse. SPLs are frequently adopted in industry by using an extractive approach, where existing system variants are the basis to create the reusable assets. The extraction of SPLs from existing variants is a complex task, because of the wide range of artifact used throughout the software engineering life-cycle and the lack of details in some of these artifacts. This situation motivates the use of SBSE to deal with SPL problems, being the extraction of SPL one of these problems. With respect to existing approaches to deal with the reengineering systems variants into SPL, we performed a systematic mapping study. As result of this mapping, we defined the reengineering process used to obtain SPLs from existing variants, we observed the limitations of existing approaches that motivate our work, and allowed us to identify related work.

Taking into account the background, related work, and limitation of existing approaches presented in this chapter, in next chapter we describe the details of our proposed approach to reengineer model variants into SPLs and to cope with the problems mentioned.

Chapter 3

The ModelVars2SPL Approach

In this chapter we describe the details of our approach called ModelVars2SPL (**Model Var**iant**s to S**oftware **P**roduct **L**ine). The goal of the approach is to provide an automated support for the entire process of reengineering UML model variants into an SPL.

In the first section we provide an overview of the approach including its steps, inputs, and outputs. Next, we devote one section for each step to present their details. Then, we describe guidelines for using the solutions obtained with ModelVars2SPL. In the last section, we recall some details of our approach.

3.1 ModelVars2SPL Overview

Figure 3.1 presents an overview of our approach to reengineer model variants into an SPL. ModelVars2SPL is composed of four steps. Each step produces different outputs that are used by other steps or are intended to support programmers, engineers and architects in the reengineering activity.

The input of ModelVars2SPL is a set of variants, as presented on the left side of Figure 3.1. Each variant consists of two parts: (i) a *UML class diagram* that provides a static view of the structure of the system variant, and (ii) a *feature set* that denotes the combination of features provided by the variant. To illustrate the input and explain the steps of the proposed approach, Figure 3.2 presents three UML class diagram[1] variants and their feature sets, which are in the captions, of a fictitious system. Regarding the two parts of each variant, for instance, Variant1 (Figure 3.2(a)) has classes `A`, `B`, `C`, and `D`; these classes have properties and methods; there are generalization and association relationships; and this variant realizes features `F1`, `F2` and `F3`. Variant2 (Figure 3.2(b)) realizes feature `F1` and `F4`, and because of its different feature set Variant2 has different model elements.

The steps of ModelVars2SPL are on the right side of Figure 3.1. ModelVars2SPL covers the entire reengineering process by having the phases of detection, analysis, and transformation, represented by dashed rectangles. The input variants are used in the three phases of the process, as pointed by the arrows. The phases of detection and analysis are composed of one step each, and transformation is composed of two steps. A brief description of each step is presented next.

Features traceability aims at identifying the model elements that implement each feature. The input for this step are both the UML class diagram and the feature set of each variant. The traceability is identified by analyzing overlaps between model elements and overlaps between

[1]The graphical representations of UML models used throughout this dissertation were obtained using the tool UML Designer: `http://www.umldesigner.org/`

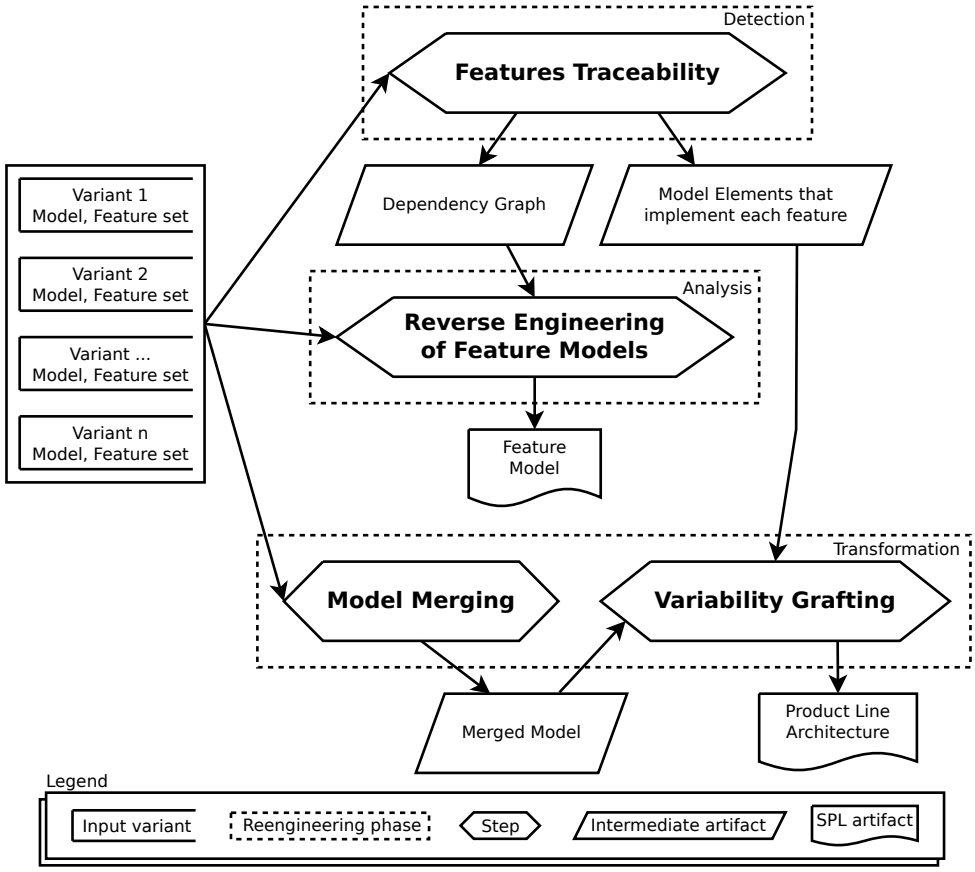


Figure 3.1: Proposed approach for reengineering systems to SPL

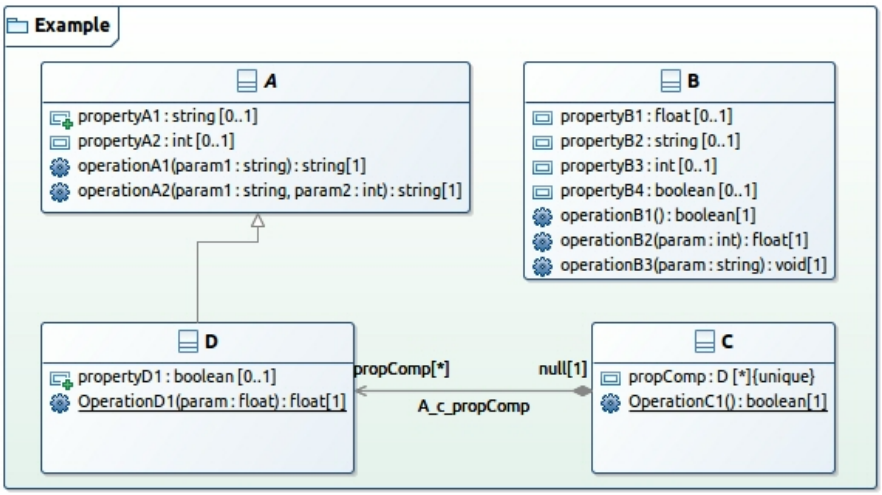
feature sets of different variants. This step produces two outputs: (i) traceability links between model elements and the features they implement, and (ii) a dependency graph that represents the relationship between features. More details of this step are presented in Section 3.2.

Reverse Engineering of Feature Models applies a multi-objective search-based technique to reach an FM that best represents the feature sets. In this step, only the feature sets of the input variants are used. Additionally, the dependency graph produced in the previous step is also used as input, with focus on generating an FM that preserves constraints of dependencies between model elements. Further details in Section 3.3.

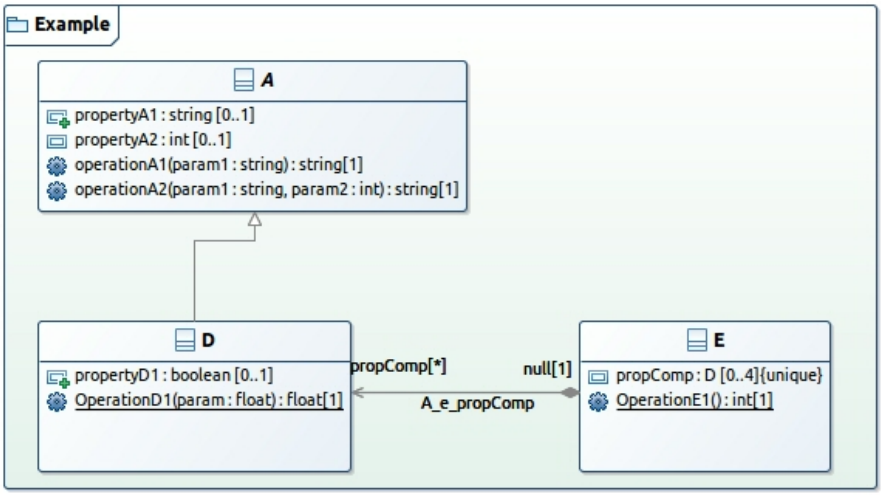
Model Merging is responsible for combining multiple class diagram variants from the input. The goal is to reach a UML class diagram with the most number of model elements spread among the model variants. This task is performed by a search-based technique that evaluates the merged class diagrams considering the number of differences from the candidate individual to all input model variants. At the end of the search process a merged model is obtained, being the candidate PLA to represent the model variants. For a complete description of this step we refer to Section 3.4.

Variability Grafting is the last step of the proposed approach. The merged model produced in previous step has only model elements (i.e., a conventional UML class diagram) without any information regarding features or variability. This step aims at enriching the conventional class diagram with information about features and variability, obtaining a PLA. Each model element of the UML class diagram is annotated with the feature it belongs. This task is done by adding a UML comment to each model element. Details are presented in Section 3.5.

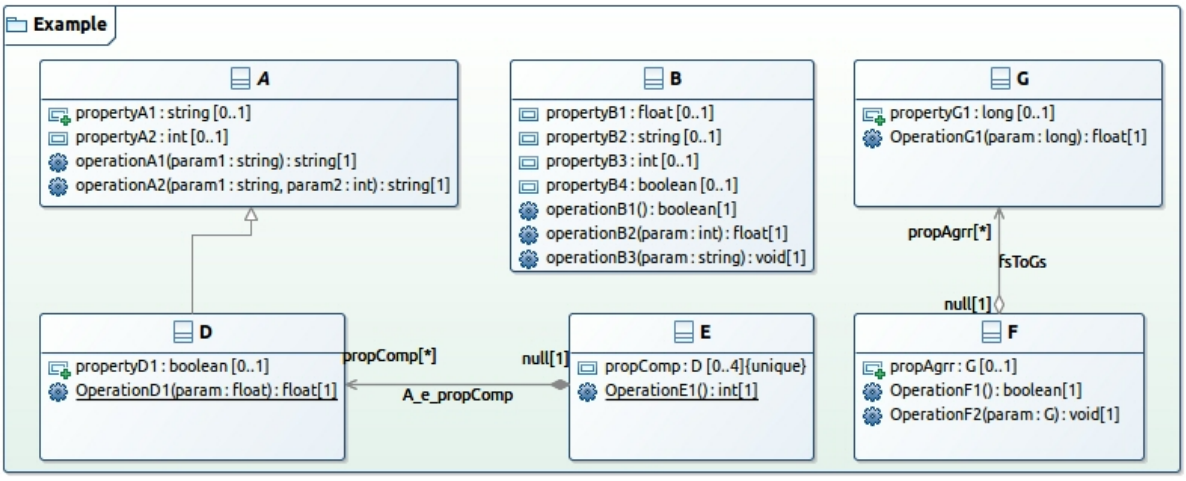
The steps of ModelVars2SPL generate five outputs, three are intermediate artifacts used as input in other step, and two are SPL artifacts. These SPL artifacts are the main output



(a) Variant1: {F1, F2, F3}



(b) Variant2: {F1, F4}



(c) Variant3: {F1, F2, F4, F5}

Figure 3.2: Example of input for the approach

artifacts of the proposed approach, they are: (i) a *Feature Model*, and (ii) a *Product Line Architecture*. Based on the input presented in Figure 3.2, Figure 3.3 presents an example of the SPL artifacts generated as outputs by ModelVars2SPL. The FM on the left side of the figure allows observing the possible combination of features and the constraints existing between them. The UML class diagram on the right side of the figure has model elements from all input variants (i.e., a complete model). Besides the complete model, this PLA has examples of feature annotations, which indicates the feature that a model element belongs. The feature annotations allow observing the traceability between features and model elements, as depicted with arrows in the figure. For example, class `A` belongs to implementation of feature `F1`, and operation `OperationC1():boolean` in class `C` belongs to feature `F3`, etc. The notation used for the annotations is described in Subsection 3.2.2.

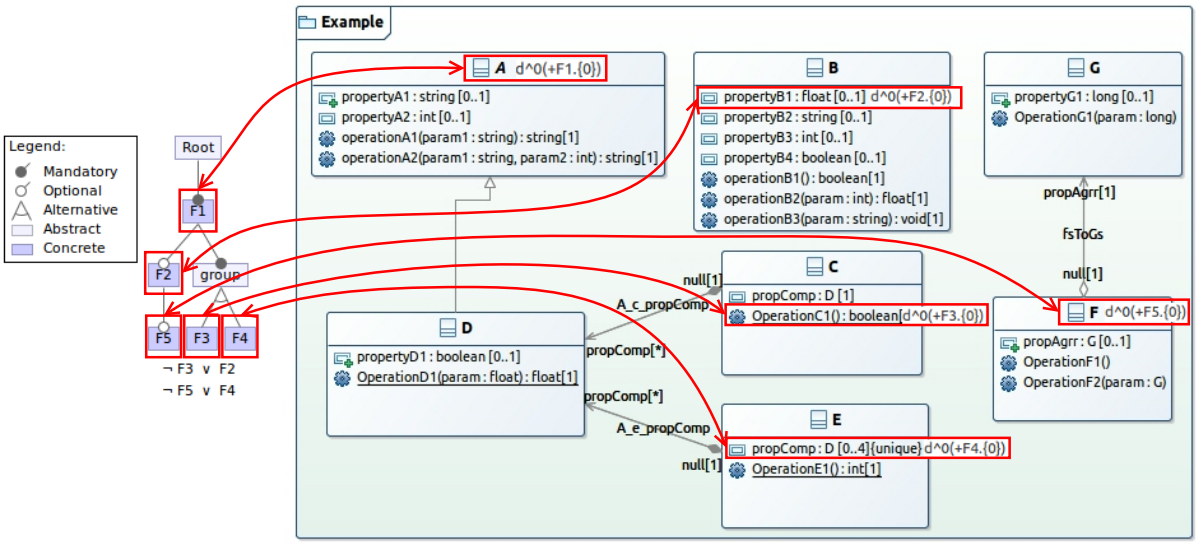


Figure 3.3: Example of output of the approach

Our approach is designed to provide a complete and automated solution for the reengineering process, but the steps can also be used individually. For instance, if an engineer needs only an FM, he/she can provide the dependency graph and the feature sets for the step of *Reverse Engineering of Feature Model* to obtain the desired FM. In another situation, if an architect wants a global view of a set of UML class diagram variants, he/she can execute only the step of *Model Merging*. This flexibility intends to make our approach suitable for different scenarios.

3.2 Features Traceability

The first step of the reengineering process aims at discovering the model elements implementing each feature. The details of this step are described in the next subsections.

3.2.1 Decomposition of Model Elements

The basis to implement the step of Features Traceability was a tool called ECCO[2]. ECCO [46] is a variability aware configuration management and revision control platform implemented in Java. This tool was initially developed to discover traceability links between

[2] `https://github.com/llinsbauer/ecco/`

features and source code [67, 68]. Currently this tool deals with text files, images, 3D models in STEP file format, Java and PHP source-code. The support for UML models is not available yet. Based on this, we developed a parser for UML models to decompose class diagrams, allowing ECCO to discover the traceability between features and model elements.

Our approach deals with models created with the Eclipse Modeling Framework (EMF)[3]. EMF is a well-known and widely used set of tools to deal with models [98]. The class diagrams in our approach are represented using EMF-based UML2[4], which is an implementation of the UML^{TM} 2.x metamodel for the Eclipse platform.

The starting point of Features Traceability is to decompose each model variant into atomic elements. Atomic elements are parts of the models with low granularity that are relevant to the reengineering process. In the parser developed to work with ECCO tool we considered as atomic model elements the class diagram types available in the UML2 EMF-based implementation aforementioned. The model element types are[5]: `ModelImpl`, `PackageImpl`, `EnumerationImpl`, `EnumerationLiteralImpl`, `ClassImpl`, `InterfaceImpl`, `PropertyImpl`, `OperationImpl`, `ParameterImpl`, `LiteralIntegerImpl`, `LiteralUnlimitedNaturalImpl`, `RedefinableTemplateSignatureImpl`, `ClassifierTemplateParameterImpl`, and `PrimitiveTypeImpl`.

A UML2 model loaded using EMF-based tools is a set of Java objects that are not serializable[6]. However, ECCO tool needs serializable objects, then we cannot use the UML2 EMF-based representation directly. We decide to represent atomic model elements using a string representation, following the pattern:

$$[Model\_Element\_Type]\ Qualified\_Name\ Additional\_Information$$

An atomic element string starts with $Model\_Element\_Type$ in brackets. $Qualified\_Name$ is composed of model name, package name, class name, etc. separated by "::". $Additional\_Information$ is not mandatory, it is used when the qualified name does not provide enough information about the element, for instance, the cardinality of a property, the type of a parameter in an operation, etc. Table 3.1 presents the atomic model elements of Variant2 (Figure 3.2(b)). Each line is an atomic element.

3.2.2 Traceability Discovery

To explain the traceability algorithm used by the ECCO tool, first we need to describe some concepts and terminology. An ideal result for the traceability discovery is to identify each set of elements that implements each distinct feature. However, in real cases, some implementation elements only appear when two features interact. These are elements responsible for providing a proper feature interaction. Furthermore, some implementation elements may appear in cases where a feature is absent. These situations should be taken into account during the traceability discovery.

Based on the possibility of features interactions and absent features, we rely on the terminology presented by Linsbauer et al. [67, 68] to distinguish two kinds of *modules* in the system variants:

---

[3] `https://eclipse.org/modeling/emf/`

[4] `http://wiki.eclipse.org/MDT/UML2`

[5] From the package: `org.eclipse.uml2.uml.internal.impl`

[6] To serialize an object means to convert its state to a byte stream so that the byte stream can be reverted back into a copy of the object.

Table 3.1: Atomic model elements of Variant2

[ModelImpl] NewModel
[PackageImpl] NewModel::Example
[ClassImpl] NewModel::Example::A
[PropertyImpl] NewModel::Example::A::propertyA1
[LiteralIntegerImpl] NewModel::Example::A::propertyA1=0
[LiteralUnlimitedNaturalImpl] NewModel::Example::A::propertyA1=1
[PropertyImpl] NewModel::Example::A::propertyA2
[LiteralIntegerImpl] NewModel::Example::A::propertyA2=0
[LiteralUnlimitedNaturalImpl] NewModel::Example::A::propertyA2=1
[OperationImpl] NewModel::Example::A::operationA1
[ParameterImpl] NewModel::Example::A::operationA1 Direction:in Name:param1 Type:string
[ParameterImpl] NewModel::Example::A::operationA1 Direction:return Name:null Type:string
[OperationImpl] NewModel::Example::A::operationA2
[ParameterImpl] NewModel::Example::A::operationA2 Direction:in Name:param1 Type:string
[ParameterImpl] NewModel::Example::A::operationA2 Direction:in Name:param2 Type:int
[ParameterImpl] NewModel::Example::A::operationA2 Direction:return Name:null Type:string
[ClassImpl] NewModel::Example::D
[PropertyImpl] NewModel::Example::D::propertyD1
[LiteralIntegerImpl] NewModel::Example::D::propertyD1=0
[LiteralUnlimitedNaturalImpl] NewModel::Example::D::propertyD1=1
[OperationImpl] NewModel::Example::D::OperationD1
[ParameterImpl] NewModel::Example::D::OperationD1 Direction:in Name:param Type:float
[ParameterImpl] NewModel::Example::D::OperationD1 Direction:return Name:null Type:float
[ClassImpl] NewModel::Example::E
[PropertyImpl] NewModel::Example::E::propComp
[LiteralIntegerImpl] NewModel::Example::E::propComp=0
[LiteralUnlimitedNaturalImpl] NewModel::Example::E::propComp=4
[OperationImpl] NewModel::Example::E::OperationE1
[ParameterImpl] NewModel::Example::E::OperationE1 Direction:return Name:null Type:int
[PackageImpl] NewModel::Common Java datatypes
...

Definition: ***Base Module.*** *A* base *module* $m = d^0(+c.\{v\})$ *implements a feature regardless of the presence or absence of any other features. This module is represented by a derivative of order 0 (*d^0*), where* $+c$ *is a feature, and* $\{v\}$ *is an optional number related to the version of the variant when the feature was included.*

Definition: ***Derivative Module.*** *A* derivative *module* $m = d^n(c_0.\{v\}, c_1.\{v\}, ..., c_n.\{v\})$ *implements feature interactions, where* c_i *is* $+F$ *(if feature* `F` *is selected) or* $-F$ *(if not selected), and n is the order of the derivative.*

ECCO tool automatically produces a set of traces between both types of modules and the implementation artifacts that implement them (i.e., atomic elements). The traces are produced by matching atomic elements overlaps and feature overlaps. Each variant is incrementally analyzed comparing overlaps already analyzed with new features and implementation artifacts provided.

To illustrate the traceability discovery we consider the input of Figure 3.2. For sake of simplicity, we consider only classes as atomic model elements. With the feature sets of each product variant and the set of atomic model elements it is possible to analyze the existing overlap between the products. This task is exemplified in Figure 3.4.

Firstly Variant1 and Variant2 are analyzed. These variants have in common the feature `F1`, Variant1 has features `F2` and `F3`, which Variant2 does not have, and this later variant has `F4`, which the former variant does not have. Regarding the model elements, both variants have

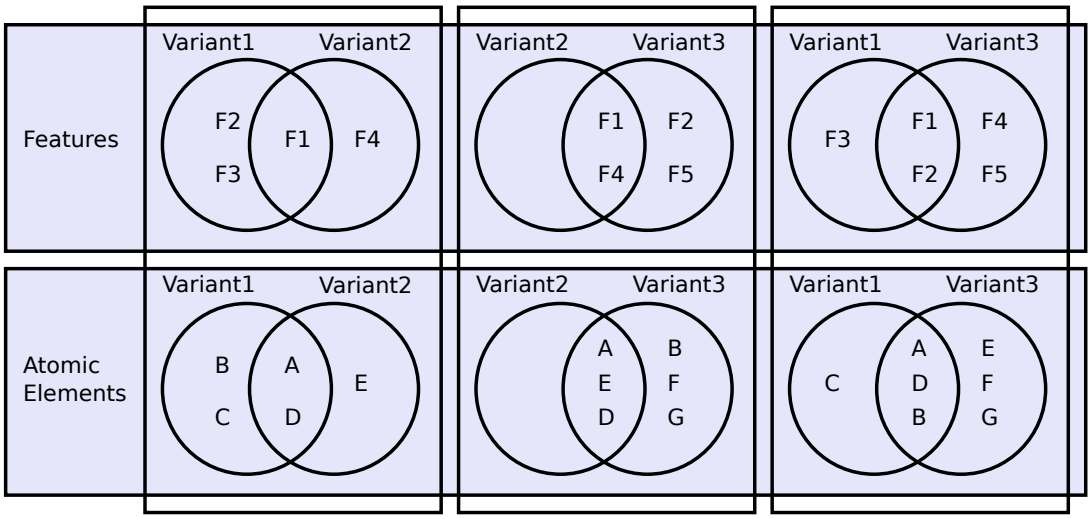


Figure 3.4: Example of overlap analysis

classes `A` and `D`, Variant1 has classes `B` and `C`, and Variant1 has class `E`. Based on this analysis we can observe that feature `F1` is implemented by model elements `A` and `D`, and feature `F4` by element `E`. For features `F2` and `F3` it is not possible to infer their implementation elements yet.

Following the same way to analyze the pair of variants, the missing traceability is discovered. For example, when analyzing Variant1 and Variant3, it becomes clear that `F3` is implemented by diagram element `C`. Then returning to the comparison results of the first analysis, now it is explicit that feature `F2` is implemented by diagram element `B`. When the analysis between all variants is finished the traceability is discovered. An excerpt of the output of the traceability discovery of our illustrated example is presented in Table 3.2.

3.2.3 Dependency Graph

Another relevant information for our approach is the relationships (e.g., dependencies) existing between elements that implement features. We obtain this information during the traceability discovery and represent it using a dependency graph. We refer to our work [8, 9] (Appendix B) to introduce the definitions regarding the dependency graph of our approach.

Definition: ***Dependency.*** *A dependency establishes a requirement relationship between two sets of modules and it is denoted with a three-tuple* `(from, to, weight)`*, where* `from` *and* `to` *are a set of modules (or module expressions) of the related modules, and* `weight` *expresses the strength of the dependency, i.e., the number of dependencies of structural elements in modules* `from` *on structural elements in modules* `to`*.*

Definition: ***Dependency Graph.*** *A dependency graph is defined as a set of dependencies, where each node in the graph corresponds to a set of modules (or module expressions), and each edge in the graph corresponds to a dependency as defined above. Edges are annotated with natural numbers that represent the dependencies' weights.*

ECCO tool automatically identifies *parent* → *child* dependencies. However, for class diagram relationships we adapted ECCO using our parser for UML2 EMF-based models. We consider in our parser the relationship types[7]: `AssociationImpl`, `UsageImpl`,

[7]From the package: `org.eclipse.uml2.uml.internal.impl`

Table 3.2: Traceability of the illustrative example

$d^0(+F1.\{0\})$ = 36 atomic elements
[ModelImpl] NewModel
[PackageImpl] NewModel::Example
[ClassImpl] NewModel::Example::A
[PropertyImpl] NewModel::Example::A::propertyA1
[LiteralIntegerImpl] NewModel::Example::A::propertyA1=0
[LiteralUnlimitedNaturalImpl] NewModel::Example::A::propertyA1=1
[PropertyImpl] NewModel::Example::A::property2
[LiteralIntegerImpl] NewModel::Example::A::property2=0
[LiteralUnlimitedNaturalImpl] NewModel::Example::A::property2=1
[OperationImpl] NewModel::Example::A::operationA1
[ParameterImpl] NewModel::Example::A::operationA1 Direction:in Name:param1 Type:string
[ParameterImpl] NewModel::Example::A::operationA1 Direction:return Name:null Type:string
[OperationImpl] NewModel::Example::A::operationA2
[ParameterImpl] NewModel::Example::A::operationA2 Direction:in Name:param1 Type:string
[ParameterImpl] NewModel::Example::A::operationA2 Direction:in Name:param2 Type:int
[ParameterImpl] NewModel::Example::A::operationA2 Direction:return Name:null Type:string
...
$d^0(+F2.\{0\})$ = 21 atomic elements
[ClassImpl] NewModel::Example::B
[PropertyImpl] NewModel::Example::B::propertyB1
[LiteralIntegerImpl] NewModel::Example::B::propertyB1=0
[LiteralUnlimitedNaturalImpl] NewModel::Example::B::propertyB1=1
[PropertyImpl] NewModel::Example::B::propertyB2
[LiteralIntegerImpl] NewModel::Example::B::propertyB2=0
[LiteralUnlimitedNaturalImpl] NewModel::Example::B::propertyB2=1
[OperationImpl] NewModel::Example::B::operationB1
[ParameterImpl] NewModel::Example::B::operationB1 Direction:return Name:null Type:boolean
[OperationImpl] NewModel::Example::B::operationB2
[ParameterImpl] NewModel::Example::B::operationB2 Direction:in Name:param Type:int
[ParameterImpl] NewModel::Example::B::operationB2 Direction:return Name:null Type:float
...
$d^0(+F3.\{0\})$ = 6 atomic elements
[ClassImpl] NewModel::Example::C
[PropertyImpl] NewModel::Example::C::propComp
[LiteralIntegerImpl] NewModel::Example::C::propComp=0
[LiteralUnlimitedNaturalImpl] NewModel::Example::C::propComp=-1
[OperationImpl] NewModel::Example::C::OperationC1
[ParameterImpl] NewModel::Example::C::OperationC1 Direction:return Name:null Type:boolean
...
$d^0(+F4.\{0\})$ = 6 atomic elements
[ClassImpl] NewModel::Example::E
[PropertyImpl] NewModel::Example::E::propComp
[LiteralIntegerImpl] NewModel::Example::E::propComp=0
[LiteralUnlimitedNaturalImpl] NewModel::Example::E::propComp=4
[OperationImpl] NewModel::Example::E::OperationC1
[ParameterImpl] NewModel::Example::E::OperationC1 Direction:return Name:null Type:int
...
$d^0(+F5.\{0\})$ = 17 atomic elements
[ClassImpl] NewModel::Example::F
[PropertyImpl] NewModel::Example::F::propAgrr
[LiteralIntegerImpl] NewModel::Example::F::propAgrr=0
[LiteralUnlimitedNaturalImpl] NewModel::Example::F::propAgrr=1
[OperationImpl] NewModel::Example::F::OperationF1
[ParameterImpl] NewModel::Example::F::OperationF1 Direction:return Name:null Type:boolean
[OperationImpl] NewModel::Example::F::OperationF2
[ParameterImpl] NewModel::Example::F::OperationF2 Direction:in Name:param Type:G
[ParameterImpl] NewModel::Example::F::OperationF2 Direction:return Name:null Type:void
...

`DependencyImpl`, `GeneralizationImpl`, `InterfaceRealizationImpl`, `AssociationClassImpl`, `TemplateBindingImpl`. For these relationship types we identify the source and target elements and describe them using the ECCO internal representation. After representing the relationships using ECCO internal types, the dependency graph is automatically generated. Figure 3.5 shows the dependency graph of our illustrative example

considering its three variants. In this example we observe only base modules (i.e., there are no derivative modules).

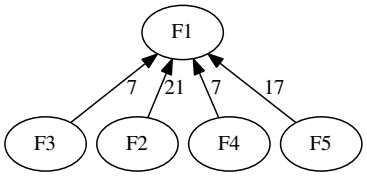


Figure 3.5: Example of dependency graph

3.3 Reverse Engineering of Feature Models

Feature Models (FMs) are the de facto standard to represent all valid combinations of features of SPLs [56]. To obtain an FM that best fits a desired set of features sets is a complex and error prone task to be performed manually. For instance, it is very frequent to derive FMs that contain more combinations than those desired by the engineer.

The step of Reverse Engineering of FMs of our approach is based on our research [8, 9] (Appendix B) that relies on a search-based technique. In these pieces of work we employ a multi-objective evolutionary algorithm to search for FMs considering three objectives, namely precision, recall, and variability safety. Two inputs are required for this task: (i) a set of feature sets from the input variants, and (ii) the dependency graph obtained in the previous step. Next we present some details based on the three ingredients described by Harman et al. [52] to implement a search-based technique (see Section 2.3): representation, fitness functions, and evolutionary operators.

3.3.1 Representation

In our search-based technique to reverse engineer FMs, the individuals are represented using a simplified version of the SPLX metamodel[8]. The metamodel used in the representation is presented in Figure 3.6. This metamodel defines the structure and semantics of FMs. The elements on the left side of the figure describe the tree-like structure of the FM, and the elements on the right side describe the cross-tree constraints (CTCs) between features. The nodes *Root*, *Mandatory*, *Optional*, and *GroupedFeature* are children of *Feature*. A tree can have only one *Root*, and the features *Mandatory* and *Optional* have the cardinality of zero or more. The number of *Alternative* and *Or* groups is arbitrary; however, each group must have at least one *GroupedFeature*. An FM has exactly one *ConstraintSet*, as presented on the right side of the figure. A *ConstraintSet* describes the propositional formula in Conjunctive Normal Form [84]. A *Constraint* has one *OrClause* that has at least one *Literal*. A literal can be an *Atom* or a *Not*.

3.3.2 Fitness Functions

Our search-based technique to reverse engineer FMs is based on a multi-objective perspective. A multi-objective evolutionary algorithm evaluates candidate solutions computing more than one fitness function. The step of reverse engineer FMs uses three fitness functions, corresponding to three optimization criteria. To describe these fitness functions, firstly we define some auxiliary functions.

[8] `http://www.splot-research.org/`

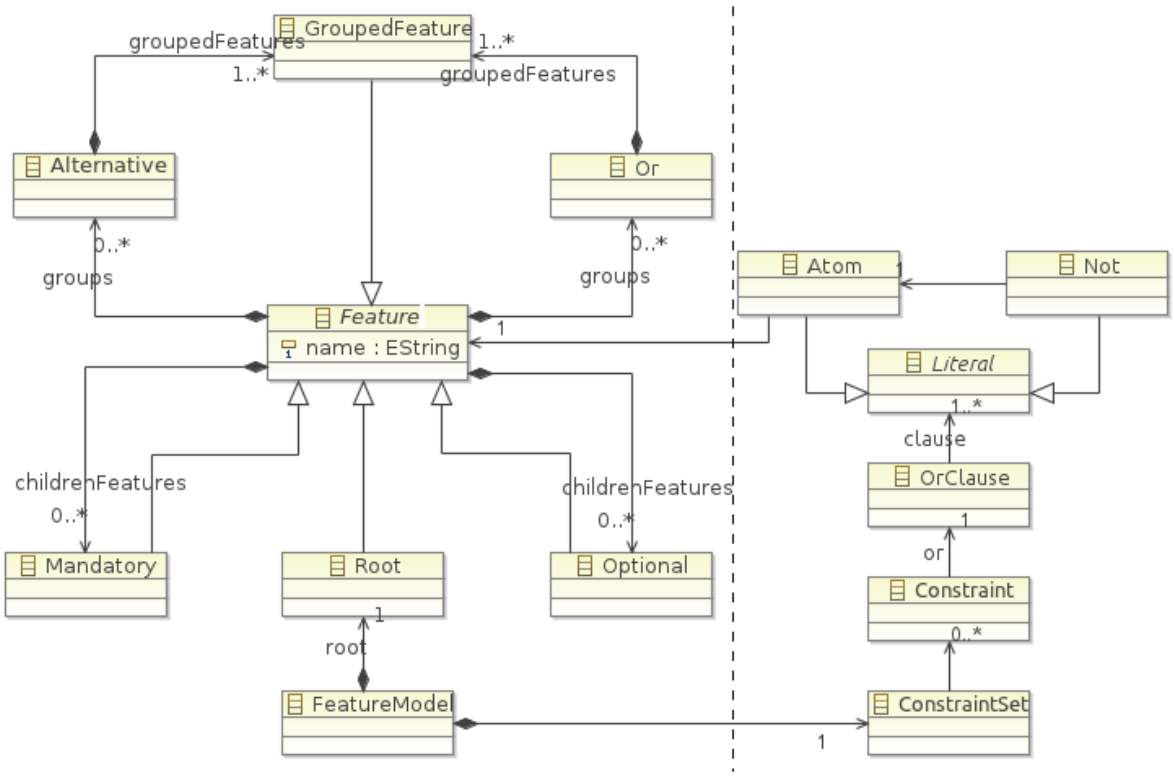


Figure 3.6: FM metamodel, extracted from [69]

Auxiliary Functions: Considering $\mathcal{FM}$ as the universe of feature models, $\mathcal{SFS}$ the universe of feature sets, and sfs the feature sets from the system variants, we define the auxiliary function called *featureSets*:

Definition: ***featureSets***. *Function* featureSets *returns the set of feature sets denoted by a feature model.*

$$featureSets : \mathcal{FM} \rightarrow \mathcal{SFS}$$

Taking into account the existence of dependencies between different features, the function *holds* checks if a dependency is not violated in a feature set. The definition of this function is:

Definition: ***holds***. *Function* $holds(dep, fs)$ *returns* 1 *if dependency dep holds on the feature set* fs *and* 0 *otherwise. A dependency dep holds for a feature set* fs *if:*

$$(\bigwedge_{f \in fs.sel} f \wedge \bigwedge_{g \in fs.\overline{sel}} \neg g) \Rightarrow (dep.from \Rightarrow dep.to)$$

Fitness Function Definitions: Considering the two auxiliary functions presented above, we introduce the three objective functions of our search-based technique. The first two functions are based on feature sets, and the third function is based on dependencies between features.

Definition: ***Precision (P).*** *Precision expresses how many of the feature sets denoted by a reverse engineered feature model* fm *are among the desired feature sets* sfs.

$$precision(sfs, fm) = \frac{|sfs \cap featureSets(fm)|}{|featureSets(fm)|}$$

Definition: ***Recall (R).*** *Recall expresses how many of the desired feature sets are denoted by the reverse engineered feature model fm.*

$$recall(sfs, fm) = \frac{|sfs \cap featureSets(fm)|}{|sfs|}$$

Definition: ***Variability Safety (VS).*** *Variability Safety expresses the degree of* variability-safety *of a reverse engineered feature model fm with respect to a dependency graph dg.*

$$variabilitySafety(fm, dg) = \sum_{dep \in dg} dep.weight \times \left(\frac{\sum_{fs \in featureSets(fm)} holds(dep, fs)}{|featureSets(fm)|} \right)$$

The three fitness functions presented above are designed to keep the computed values between 0 and 1. Moreover, the goal is to maximize the values for all functions. An ideal solution is P=1.0, R=1.0, and VS=1.0.

3.3.3 Evolutionary Operators

The evolutionary operators of crossover and mutation used in this step are restrict to some domain constraints. To guarantee the semantics of FMs and to avoid invalid solutions the domain constraints are as follows:

- Each feature is identified by its name, so every feature appears exactly once in the FM tree;
- All FMs have a fixed set of feature names, so in different FMs only the relations between features are different;
- CTCs can only be either *requires* or *excludes*, i.e., exactly two literals per clause with at least one being negated;
- CTCs must not contradict each other, i.e., the corresponding CNF of the entire constraint set must be satisfiable;
- There is a maximum number of CTCs (given as a percentage of the number of features) that must not be exceeded.

Crossover: The operator of crossover generates offspring following the metamodel representation and in conformance to the domain constraints. The crossover process is:

1. Initialize the offspring with the root feature of $Parent_1$. If the root feature of $Parent_2$ is a different one then it is added to the offspring as a mandatory child feature of its root feature.
2. Traverse the first parent depth first starting at the *root* node and add to the offspring a random number r of features that are not already contained by appending them to their respective parent feature already contained in the offspring using the same relation type between them (the parent feature of every visited feature during the traversal is guaranteed to be contained in the offspring due to the depth first traversal order).
3. Traverse the second parent exactly the same way as the first one.
4. Go to Step 2 until every feature is contained in the offspring.

The second child is obtained by swapping the order of the parents. The CTCs offspring are assigned randomly from a set of constraints obtained by merging all the constraints of both parents. First a subset of CTCs are selected and assigned to the first offspring and then the remaining to the second offspring.

Mutation: The mutation operator applies minor modifications in the tree or in the CTCs of the FM. The mutation probability is used to choose if the change is applied in the tree part, in the CTCs, or in both. The modifications applied in an FM are:

- Mutations performed on the tree:
  - Randomly swaps two features in the feature tree;
  - Randomly changes an *Alternative* relation to an *Or* relation or vice-versa;
  - Randomly changes an *Optional* or *Mandatory* relation to any other kind of relation (*Mandatory*, *Optional*, *Alternative*, *Or*);
  - Randomly selects a subtree in the feature tree and puts it somewhere else in the tree without violating the metamodel or any of the domain constraints.
- Mutations performed on the CTCs:
  - Adds a new randomly created CTC that does not contradict the other CTCs and does not already exist;
  - Randomly removes a CTC.

Selection: The individuals chosen to participate in the crossover/mutation are selected by standard tournament selection as selection operator. In this strategy a set of individuals are randomly selected from the population, then the individual with the best fitness is selected.

Initial population: Following the representation described above, the initial population is created by generating random feature trees and random CTCs. Some domain constraints, which will be presented next, are also taken into account to generate random FMs. The random FMs are created using the tools FaMa [19] and BeTTy [94].

3.3.4 Output

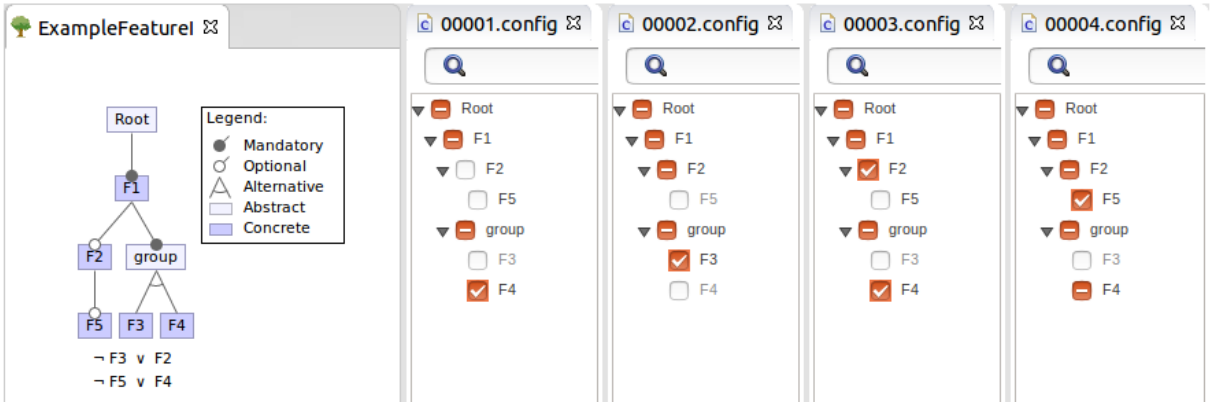
To illustrate the reverse engineering of FMs we recall the input of Figure 3.2. The set of feature sets for our illustrative example is presented in Table 3.3. The dependencies of this illustrative example is presented in Figure 3.5, however we represented the dependency graph with normalized values in Table 3.4.

Table 3.3: Feature sets of the illustrative example

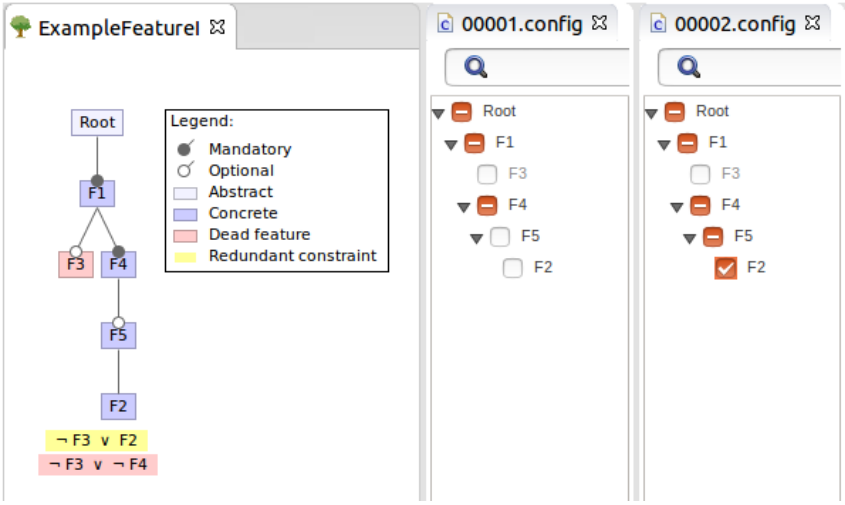
Variants	**Feature Sets**				
	F1	F2	F3	F4	F5
Variant1	✓	✓	✓		
Variant2	✓			✓	
Variant3	✓	✓		✓	✓

Table 3.4: Dependency matrix of the illustrative example

From	To	Weight	Normalized
F2	F1	21	0.4038461538
F3	F1	7	0.1346153846
F4	F1	7	0.1346153846
F5	F1	17	0.3269230769
Total:		161	1.0000



(a) FM1: P=0.75, R=1.0, VS=1.0



(b) FM2: P=1.0, R=0.67, VS=1.0

Figure 3.7: Example of reverse engineered FMs

The information of Tables 3.3 and 3.4 were provided as input for the NSGA-II (Non-dominated Sorting Genetic Algorithm) [34] using our implementation reported in [8, 9] (Appendix B). The solutions reached by the algorithm are presented in Figure 3.7. The FMs and configurations presented in this figure are created using the tool FeatureIDE[9]. In a brief analysis we can observe that both FMs have VS equal to 1.0, which means they do not violate any dependency of the dependency graph. Regarding the objectives of precision and recall, there is a trade-off between the solutions. FM1 (Figure 3.7(a)) has R=1.0, which means the three configurations of Table 3.3 are possible with this FM, however, the same FM denotes more

[9] http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/

products than desired, decreasing the values of precision to 0.75. On the other hand, the value of precision for FM2 (Figure 3.7(b)) is 1.0, therefore all products of this FM are in the desired feature sets, however it denotes only two products, missing one desired configuration, leading to a recall value equal to 0.67. The software engineer can choose the FMs that best fits his/her intentions. With this example we observe the benefits on having a multi-objective solution.

3.4 Model Merging

Model merging is the first step of the transformation phase of the reengineering process. The goal of this step is to generate a global class diagram with the highest similarity to the input variants. To reach this highest similarity, the global class diagram must have as many as possible the model elements spread among the variants. The basis for this step is our work [11] (Appendix C). In this work, we applied a search-based technique that considers differences among the global class diagram and then find a class diagram that has the most similar structure with all the model variants considered as input. In this section we present details of our search-based technique to discover software architectures, again, based on the three ingredients of SBSE described by Harman et al. [52] (see Section 2.3).

3.4.1 Representation

The representation of the UML models in this step uses the same structure of the step Features Traceability (Section 3.2). The class diagrams are represented using EMF-based UML2 implementation, described using the Ecore metamodel. Figure 3.8 presents the components, relations, attributes, and operations of the Ecore metalmodel[10]. When models are represented using this metamodel, they can be compared, modified, and saved. These operations enabled by EMF tools are the basis to our search-based technique.

3.4.2 Fitness Function

The fitness function of our technique is based on differences among UML class diagrams of the system variants. To compute these differences we use the Eclipse EMF Diff/Merge tool[11]. EMF Diff/Merge compares two models and returns the differences between them. EMF Diff/Merge computes three essential types of differences between models: (i) presence of an unmatched element, which refers to an element in a model that has no match in the opposite model; (ii) presence of an unmatched reference value, which means that a matched element refers another element in only one model; (iii) presence of an unmatched attribute value, where a matched element owns a certain attribute value in only one model.

EMF Diff/Merge tool is able to compare only two at once (or three models if we include an ancestor model). However, to evaluate a candidate architecture we have to compute differences from one UML class diagram to many UML class diagram variants. Considering this, the proposed fitness function is composed of the sum of differences from one model to all input model variants. The definition below presents the fitness function called *Model Similarity*. The function *diff* represents the number of differences found by using EMF Diff/Merge. We sum only the set of differences indicating that the elements exist in the variant *v* but are missing on the *candidate_model*.

[10] `http://download.eclipse.org/modeling/emf/emf/javadoc/2.11/org/eclipse/emf/ecore/package-summary.html`

[11] `http://www.eclipse.org/diffmerge/`

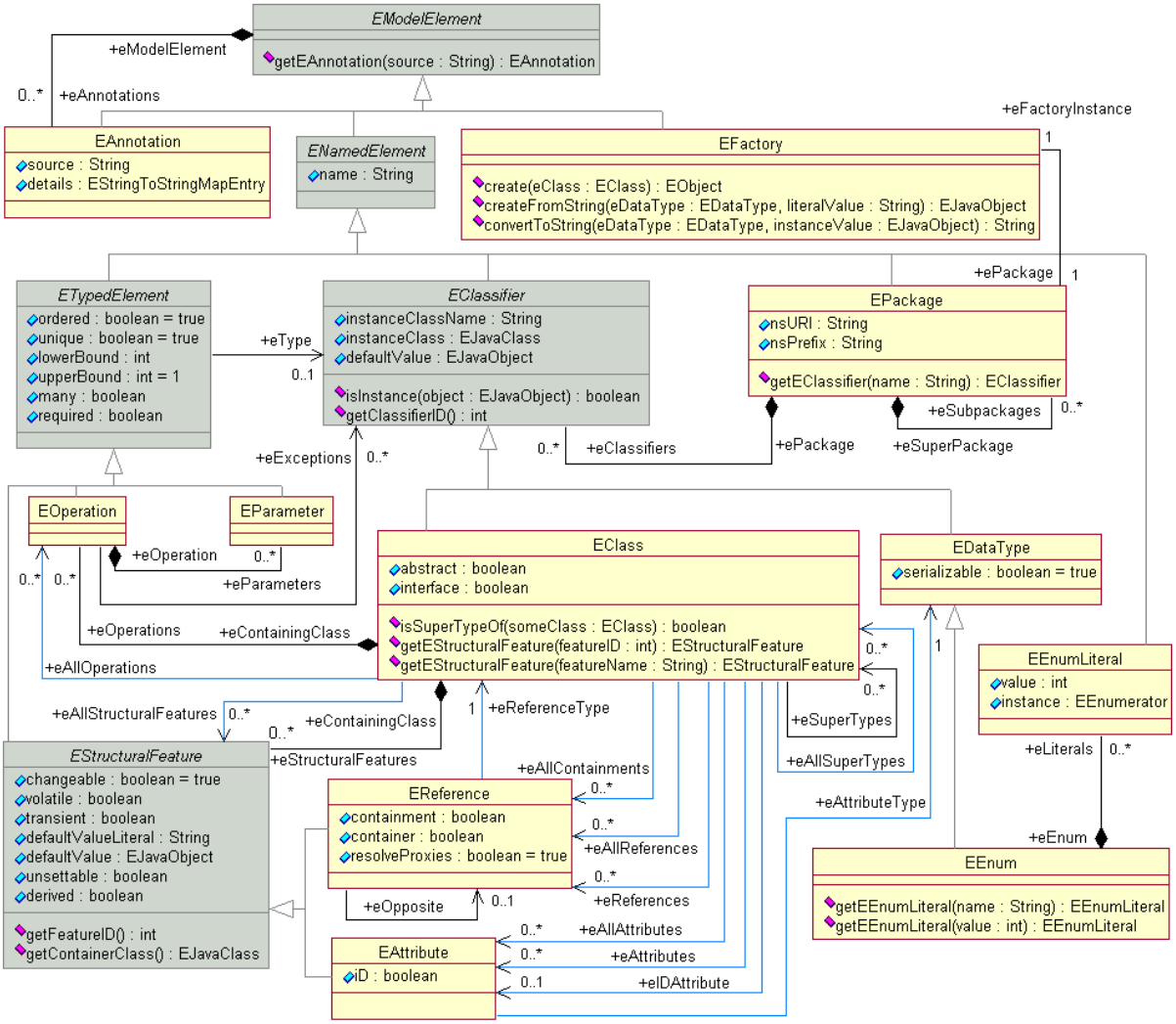


Figure 3.8: Ecore metalmodel, extracted from EMF documentation

Definition: ***Model Similarity (MS).*** *Model Similarity expresses the degree of similarity of the candidate architecture model to a set of model variants.*

$$MS = \sum_{v \in Variants} diff(candidate\_model, v) \tag{3.1}$$

3.4.3 Evolutionary Operators

The set of differences returned by EMF Diff/Merge tool is used to perform crossover and mutation. EMF Diff/Merge tool also allows duplication/modification of models to incorporate the changes done by the operators.

Crossover: The starting point of our crossover operator is two candidate architectures. From these two models we generate two children: one with the differences merged and one without the differences. For instance, let us consider two parent models X and Y. The children will be:

- *Crossover Child Model 1*: this model has the differences between its parents merged. For example, the elements of X that are missing on Y are merged in this latter, or vice versa. Both ways will produce the same child.
- *Crossover Child Model 2*: this child is generated by removing the differences between the parents. For example, the differences of X that are missing on Y are removed, or vice versa. Both ways will produce the same child.

The strategy adopted to *Child Model 1* aims at creating a model that has more elements, going to the direction of the system architecture. On the other hand, the strategy used for *Child Model 2* has the goal of eliminating possible conflicting elements from a candidate architecture.

Mutation: Mutation operator aims at applying only one modification in each model parent. The starting point of mutation operator is two candidate architectures, and the result is also two children. Let us consider any parent models X and Y. The children are:

- *Mutation Child Model 1*: the first child is created by merging one difference of the model Y in the model X. After randomly selecting one element of the model Y, but missing on the model X, this element is added in the model X.
- *Mutation Child Model 2*: here, the same process described above is performed, however, including one element of the model X in the model Y.

The mutation process can select a difference that is owned (i.e., is part of) another difference. In such cases, the entire owning difference is moved to the child. For example, when a mutation selects a parameter owned by an operation, the entire operation is moved to the child.

Selection: The selection of solutions to apply crossover/mutation is done by the tournament strategy.

Initial Population: In our approach the initial population is created by copying the input UML models variants. The only constraint is that all the input models should be included in the initial population at least once. When the population size is greater than the number of input variants, multiple copies of each variant are added in the initial population.

3.4.4 Output

An example of output generated by our model merging approach is presented in Figure 3.9, considering the three variants of the illustrative example (Figure 3.2). We can observe that all model elements of the input variants are merged in one global UML class diagram. This merged model is the starting point to obtain the PLA.

3.5 Variability Grafting

In the context of this dissertation, a PLA is a structural representation of model variants with annotations regarding the variabilities and features of the variants. The structural representation is provided by the merged model reached in the previous step. In this step the traceability

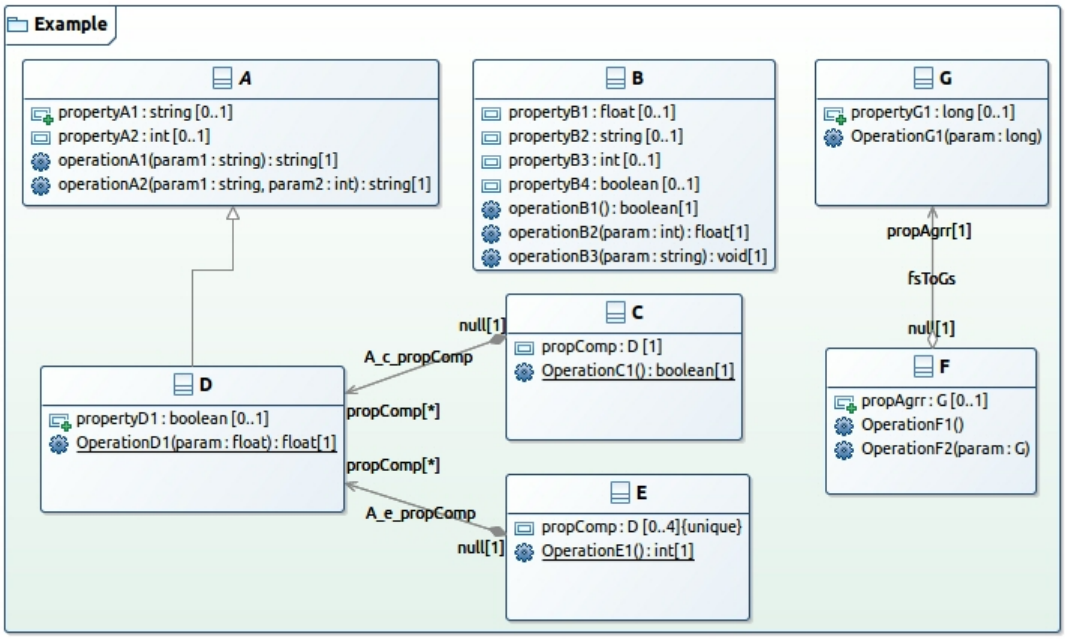


Figure 3.9: Example of merged UML class diagram

discovered in the step Features Traceability (Section 3.2) is used to graft information regarding the features in the merged model.

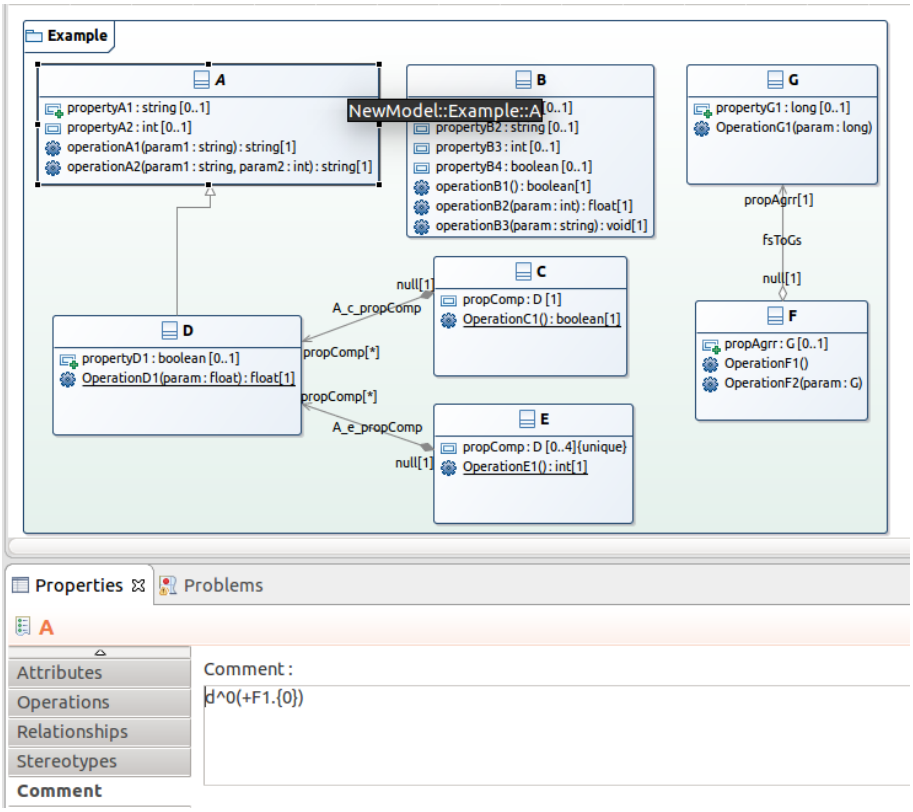
The task of variability grafting means indicating to which module (i.e., features) each model element belongs. This information supports the migration from ad hoc reuse to reengineering the implementation artifacts to an SPL platform. Besides supporting the migration, this PLA provides information to programmers, engineers, and architects to easily identify the implementation artifacts in the UML class diagram that are responsible to implement a feature, easing the maintenance. Moreover, knowing the module of each model element, the programmers, engineers, and architects can make a better decision regarding the evolution of the system, by adding more products, or improving the internal quality of the software, because of the provided global view of the system.

For this step we decided to adopt a way to allow having a good view of an architecture. The information of variability is graft in the merged model by using UML owned comments. Each element in the UML class diagram can have owned comments. Adopting this strategy of owned comments, the PLA can be viewed in any UML editor.

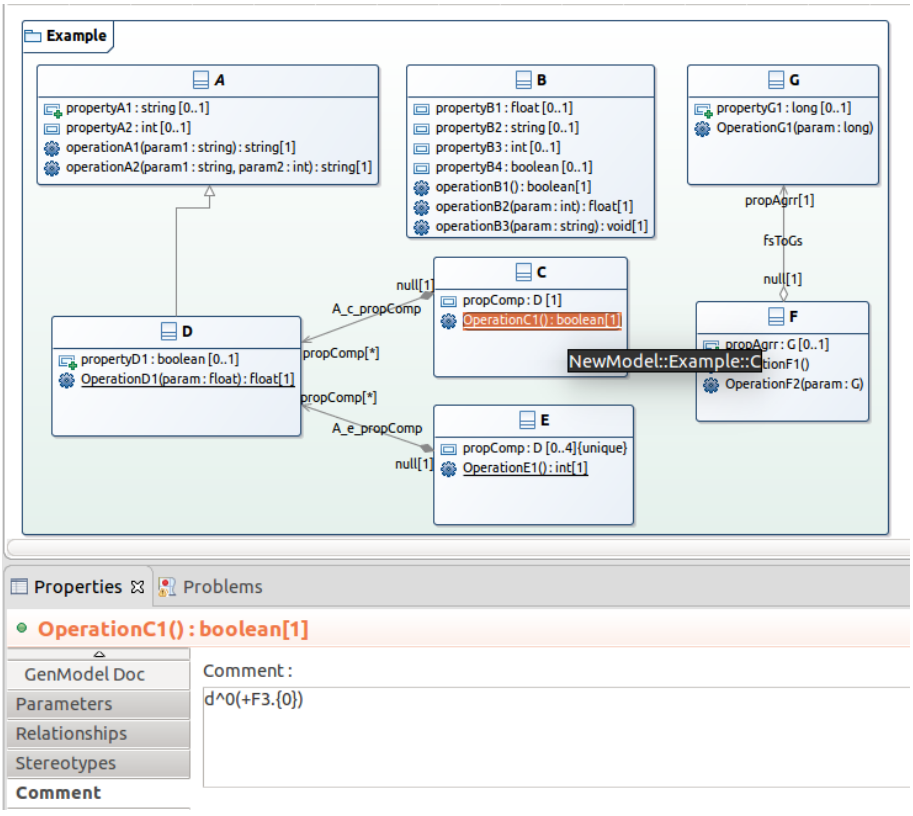
Figure 3.10 presents a PLA constructed using the merged model presented in Figure 3.9 and the traceability presented in Table 3.2. In Figure 3.10(a) the class `A` at the top of the figure is selected, and at the bottom we can see that this model element belongs to the module $d^0(F1)$. Figure 3.10(b) presents the variability information of an operation of `C` with the comment that indicates it belongs to $d^0(F3)$.

3.6 Guidelines for Using ModelVars2SPL Solutions

This section is devoted to give a practical overview on how to use the solutions obtained by ModelVars2SPL. We focus on the activities of maintenance, evolution, and testing, but the use of the solutions are not limited only to these activities.



(a) Comment of Class with F1



(b) Comment of Operation with F3

Figure 3.10: Example of variabilities in the PLA

Maintenance: Maintenance is a challenge activity in the context of ad hoc reuse, as we discussed in Section 2.1. However, this activity may become easier when adopting an SPL. Next we present some situations where the FM and the PLA can help the maintenance activity of many systems.

- *Bug fixing*: the solutions of our approach put together implementation elements that were spread over different variants. Once we have the global model (i.e., the PLA), the bugs found during the use of the system variants can be fixed and the correction will be present in all products variants instantiated from the SPL.
- *Detection of bad smells and Refactoring*: the implementation elements spread over many variants are difficult to be analyzed and the improvement of the internal quality of those systems becomes a complex activity. Using a PLA, the model elements of many variants are all in the same place, easing the visualization and analysis to find bad smells and proposing the corresponding refactoring operations. By the use of an FM, we can analyze the different combinations of features and identify possible critical behavior for a specific combination.

Evolution: Below we provide some points with respect to the use of the solutions of our approach for the evolution activity.

- *Production of new products*: by the use of FMs and PLAs, programmers, engineers, and architects have a global view of existing system variants. From this global view, practitioners can make decisions regarding the growing of the SPL, producing new variants with the existing features or measuring the impact of adding new features in the SPL.
- *Variability management*: The PLA representation of a set of variants helps to understand how the variable and common parts of the variants are implemented. Using this knowledge, programmers, engineers, and architects can improve the design, allowing better reuse of existing parts in new products or separating implementation elements of different features. For instance, to separate hardware aspects from functionalities in order to create products for new hardware (e.g., migration to a mobile platform).
- *Improve performance*: the global visualization provided by the FM and PLA may help practitioners to improve non-functional requirements by putting together different implementations and by allowing deep analysis of variants behavior.

Testing: The test activity of many systems is a complex task because the implementations elements are spread over the variants. The use of an FM and a PLA may benefit the testing activity by:

- *Design of test cases*: our approach makes explicit the interactions between features and the corresponding model elements that interact. Therefore, the PLA and FM support the design of test cases to test feature interactions, easing integration testing.

3.7 Concluding Remarks

ModelVars2SPL is an approach to reengineer model variants into SPLs. The input of the approach is a set of UML class diagram variants, and a set of features sets describing the

features of each variant. ModelVars2SPL covers the entire reengineer process, being composed of four steps: (i) *Features traceability* discovers the implementation elements of the features; (ii) *Reverse Engineering of FMs* generates FMs that represent the combination of features and constraints existing between features in the input variants; (iii) *Model Merging* takes all input class diagrams and creates a global architecture; and (iv) *Variability Grafting* inserts annotations in the global architecture to describe the features related to each model element. Each step has its own technique, being the Features traceability and Variability Grafting based on deterministic algorithms, and Reverse Engineering of FMs and Model Merging performed by search-based techniques.

The next chapter describes an experimentation of the proposed approach to evaluate the quality of the solutions obtained with ModelVars2SPL.

Chapter 4

Evaluation of the Proposed Approach

In this chapter we describe the details of the evaluation of ModelVars2SPL. The goal is to conduct an experiment to evaluate the quality of the solutions obtained by the proposed approach. Based on this, first we present the research questions to be answered with the experimentation. In the second and third sections we discuss how we measure the quality of the obtained solutions to answer the research questions. Next the experimental settings are described. Then results and analysis are presented for each step of the approach and a discussing is given. An example is presented to illustrate the use of the solutions obtained. The last sections are devoted to threats to validity and concluding remarks.

4.1 Research Questions

To evaluate ModelVars2SPL we conducted an experimentation with the goal of obtaining some findings related to the quality of the solutions. To this end, two research questions were established to guide our experimentation, as follows:

RQ1: *How is the quality of the FM and the PLA obtained with ModelVars2SPL in comparison to the input variants?* The proposed approach is designed to obtain FMs and PLAs that best represent a set of input variants. However, in practice the input for the approach may be composed of few variants implementing the existing features, or many variants implementing the system features. With this research question we want to investigate the behavior of ModelVars2SPL for different input scenarios regarding the number of features implemented in the input variants. To answer this research question our analysis relies on attributes of correctness, which are described in Section 4.2.

RQ2: *How is the quality of the intermediary output artifacts generated in each step of the approach?* As described in Chapter 3, our approach is composed of deterministic and search-based techniques. These two kinds of techniques have different characteristics and should be evaluated properly. This research question aims at investigating whether the solutions obtained in each step of ModelVars2SPL are according to the goal of each phase of the reengineering process. To answer this research question we consider measures specific for the technique used in each step. In addition, we present and illustrate the use of the intermediary outputs for one case study. These measures are described in Section 4.3

4.2 Measures to Evaluate the FM and the PLA

In our systematic mapping we observed a lack of measures to evaluate reengineered SPLs [10]. Based on this, we used an existing work on the quality of FMs as basis to create our reengineered-based measure to evaluate the quality of the FM and PLA obtained with ModelVars2SPL.

Thörn proposed a quality model composed of six factors and 25 attributes to evaluate the quality of FMs [102]. Based on the scope of this dissertation, among the six quality factors we focus on the factor of correctness, also called veracity. According to the author *correctness*, or *veracity*, "*is the quality factor that relates to how well the model corresponds to the real world, i.e., it is the degree of precision in mapping the modeled product line to the abstractions in the model*". Correctness is affected by five attributes: (i) *accuracy* describes how well the model represents the actual world; (ii) *redundancy* measures how concise is the model on providing the smallest representation of the SPL, this attribute is similar to our recall measure used in the step of Reverse Engineering of FMs (see Section 3.3); (iii) *completeness* is related to how well a model covers the product set that constitutes the SPL, this attribute is similar to our precision measure used in the step of Reverse Engineering of FMs (Section 3.3); (iv) *reliability* represents the level of confidence that an FM will result in configurations that will perform their intended functionalities; and (v) *robustness* denotes the resistance of the model to result in incorrect configurations, for example, describing domain constraints.

Despite of correctness be designed for evaluating the quality of FMs, which is one of the outputs of our approach, we extended its applicability to encompass the measurement of PLAs. Moreover, differently from introduced by Thörn, we use another baseline to perform the comparisons. Taking into account that the main goal of our reengineering approach is to represent a set of input model variants into an SPL, then instead of considering the real world to evaluate the correctness of the FM and the PLA, we can use these input variants as the baseline for comparisons. In other words, we use known variants to analyze the five quality attributes of correctness for the solutions obtained with ModelVars2SPL.

4.3 Measures to Evaluate the Outputs of Each Step

Besides the FMs and the PLAs obtained with ModelVars2SPL, our approach generates intermediary output artifacts in each step. Here we present the measures we used to evaluate these intermediary outputs. Considering the steps use different kinds of techniques, some measures are used in specific steps, as described next.

Runtime: During the execution of each step of our approach we measured the time performance. For the steps using exact techniques (deterministic algorithms), namely Features Traceability and Variability Grafting, the runtime of each single run for each case study was collected. For the steps using search-based techniques (stochastic algorithms), namely Reverse Engineering of FMs and Model Merging, we computed the mean runtime for the 30 independent runs.

Pareto Front: The second step of our approach deals with the problem of reverse engineering FMs as a multi-objective optimization. A multi-objective problem usually does not have a single solution, since it depends on diverse factors (objectives) that are in conflict. Hence, diverse good solutions might be possible. One of the solutions could be better considering one or more

objectives, and other solutions considering the remaining objectives. These solutions are called non-dominated and form the Pareto front [81]. To evaluate the step of Reverse Engineering of FMs we constructed the Pareto Front based on non-dominated solutions reached considering the 30 independent runs.

Euclidean distance from an ideal solution (ED): ED is a quality indicator with the purpose of finding the closest solutions to the best objectives (i.e., the ideal solution). An ideal solution has the best value of each objective [27]. ED helps to select solutions from the Pareto Front, since it indicates the solution that has a good trade-off among all objectives. We used ED to evaluate solutions of the Pareto Front obtained in the step Reverse Engineering of FMs. ED is commonly used in SBSE [105].

Evolution of the best solution: To evaluate the ability of our search-based technique to merge different models, we considered the evolution of the best solution during the evolutionary process. At the end of each generation, we collected the fitness value of the best solution. Based on the analysis of the best solutions in all generations we can observe if the search-based algorithm is able to explore the search space and reach good solutions.

Illustration of use: During the analysis of the results in each step, we also present and illustrate the use of the intermediary artifacts obtained for the Banking System case study (see Section 4.4). We choose this case study mainly because of its size and possibility of displaying their models.

4.4 Case Studies

In order to answer the research questions, we selected case studies based on two scenarios. We assume that for each case study we have a set of variants implementing a set of features and that these features can be combined to generate variants. Based on this, the scenarios are:

- *Scenario 1*: In this scenario the case studies are composed of variants which implement all possible configurations (i.e., combinations of features). For instance, there are variants which implement few features and a variant which implements the maximum possible number of features. This scenario helps us to evaluate the effectiveness of our approach in obtaining the same products of an SPL in comparison with the initial individual variants used in our approach.

- *Scenario 2*: The case studies in this scenario are composed of variants with the combinations of at most half of the features. To select the variants of Scenario 2 we used the same rule of our previous work on architecture recovery [11] (Appendix C), as follows:

$$threshold = \left\lceil \frac{\#all\_features - \#mandatory\_features}{2} \right\rceil + \#mandatory\_features$$

  Only the variants that have the number of implemented features below the threshold were selected. When some feature of the case study was not present in any of the variants, then we selected one additional variant to have the implementation of that feature at least once. The goal of case studies in Scenario 2 is to evaluate the effectiveness of our approach on merging models to obtain one global model with all implementation elements spread over different variants.

In our experimentation we used ten case studies, five case studies in each scenario. Each case study is a set of different UML class diagram variants where each variant implements a set of different features, and is composed of classes, attributes, operations and relationships.

The case studies are: Banking System (BS), Draw Product Line (DPL), Mobile Media (MM), Video On Demand (VOD), ZipMe (ZM), and Game Of Life (GOL). BS is a small banking application composed of four features [74]. DPL is a collection of products to draw lines and rectangles and has six features [8]. MM is an application to manipulate media files, such as photo, music, and video, on mobile devices [45]. For the case study MM we used five versions, namely MMv1 to MMv5. VOD implements eleven features for video-on-demand streaming [8]. ZM is a set of tools, with seven features, for files compression [8]. GOL is a customizable game [8].

Table 4.1 presents the total number of features and variants, and the mean number (and standard deviation) of classes, interfaces, attributes, operations, and relationships. To compute the information regarding classes, attributes, operations and relationships we used SDMetrics[1]. We reverse engineered the models from Java code using Eclipse MoDisco[2], except the BS case study which was originally a set of UML model variants. The detailed information of all variants for each case study is presented in Appendix D.

Table 4.1: Case studies

Case Study	#F	#V	Mean (Std Deviation)				
			Cl	Int	Attr	Op	Rel
Scenario 1							
BS	4	8	4.00 (0.76)	0.00 (0.00)	7.00 (1.31)	10.00 (2.93)	2.88 (1.36)
DPL	6	16	4.69 (1.25)	0.00 (0.00)	13.50 (5.39)	35.44 (10.25)	5.13 (1.63)
MMv1	5	3	23.00 (0.00)	1.00 (0.00)	22.00 (0.00)	121.00 (0.00)	13.00 (0.00)
VOD	11	32	37.00 (3.05)	0.00 (0.00)	280.00 (0.00)	233.00 (9.34)	91.50 (7.89)
ZM	7	32	25.00 (1.76)	3.00 (0.00)	193.00 (6.10)	283.50 (23.54)	74.75 (5.83)
Scenario 2							
GOL	15	28	15.61 (1.26)	1.96 (0.19)	22.79 (2.78)	74.57 (5.07)	32.32 (1.31)
MMv2	6	4	24.00 (0.00)	1.00 (0.00)	25.25 (0.50)	134.25 (2.50)	21.00 (0.00)
MMv3	7	9	24.00 (0.00)	1.00 (0.00)	25.67 (0.50)	135.67 (2.18)	21.00 (0.00)
MMv4	8	12	28.25 (0.45)	1.00 (0.00)	25.75 (0.45)	150.00 (1.95)	25.75 (1.36)
MMv5	9	33	31.18 (3.72)	1.00 (0.00)	32.55 (8.43)	167.64 (19.37)	27.45 (2.05)

#F: Features, #V: Variants, Cl: Classes, Int: Interfaces, Attr: Attributes, Op: Operations, Rel: Relationships

Figure 4.1 provides an overview regarding the number of features and model elements in each variant of the case studies. We can see that BS variants have a similar number of attributes and operations. In the variants of all MM versions, DPL, GOL, and ZM the majority of model elements are also operations. VOD variants have numerous attributes. VOD, ZM, and GOL have the greates number of relationships.

For sake of illustration, Figure 4.2 presents all the UML class diagram variants of BS. As mentioned before, the BS case study has four features and is composed of eight variants. BS1 (Figure 4.2(a)) is the simplest variant, which implements only the common operations of the banking system. BS8 (Figure 4.2(h)) is the most complete variant implementing the four features.

[1]http://www.sdmetrics.com

[2]https://eclipse.org/MoDisco

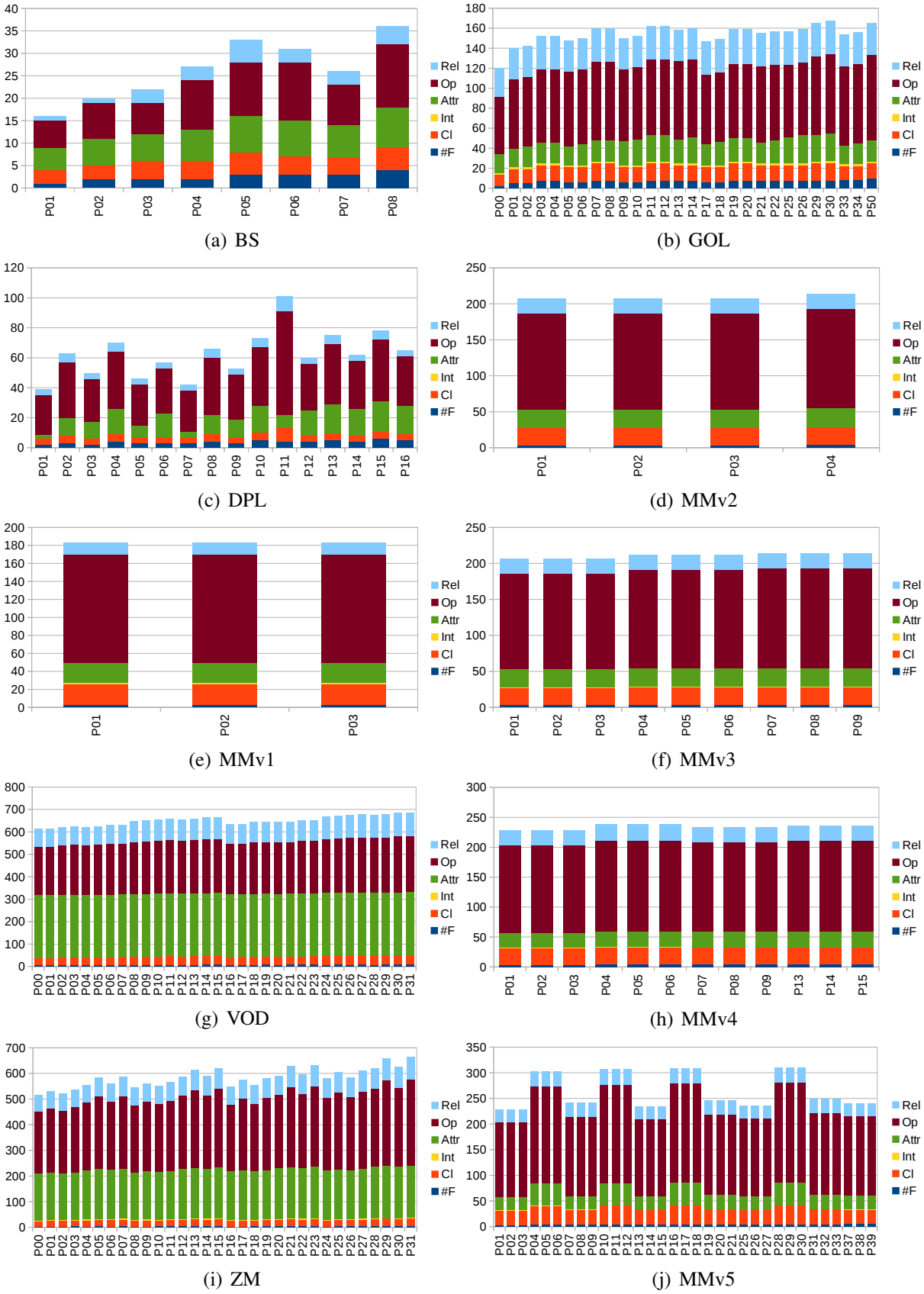
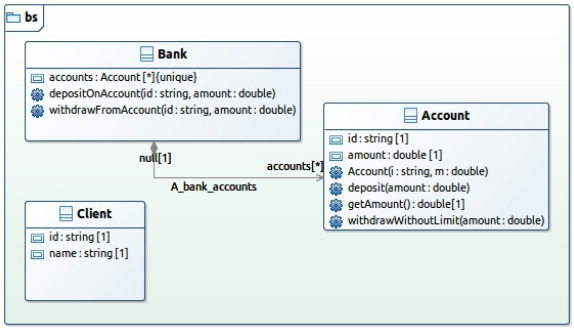
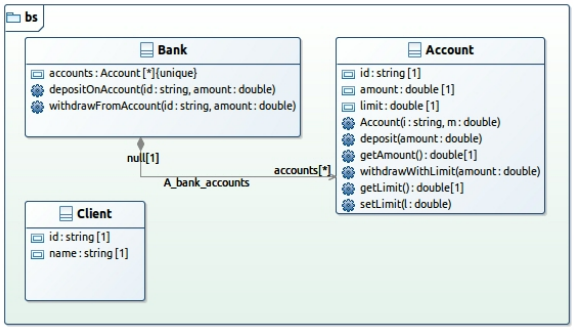


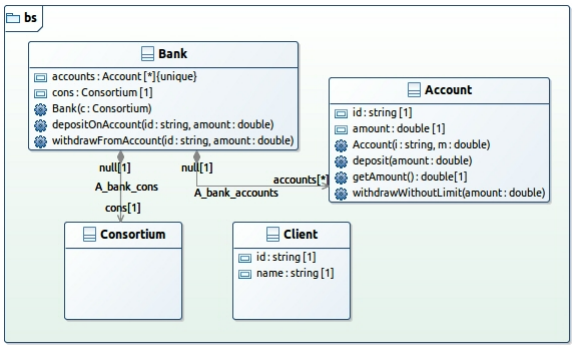
Figure 4.1: Model elements of the variants in each case study



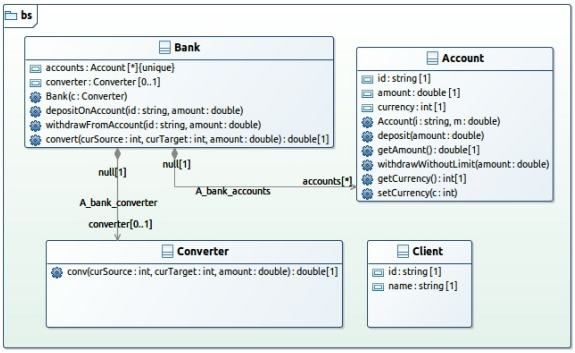
(a) BS1: {Base}



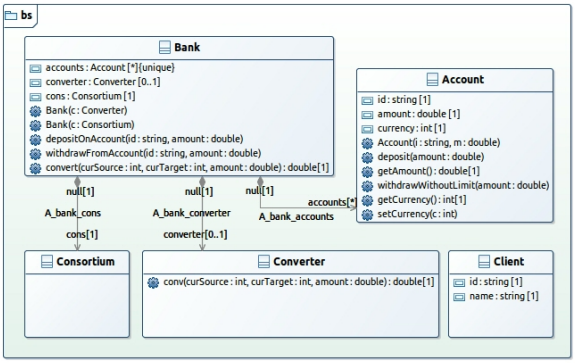
(b) BS2: {Base, WithdrawLimit}



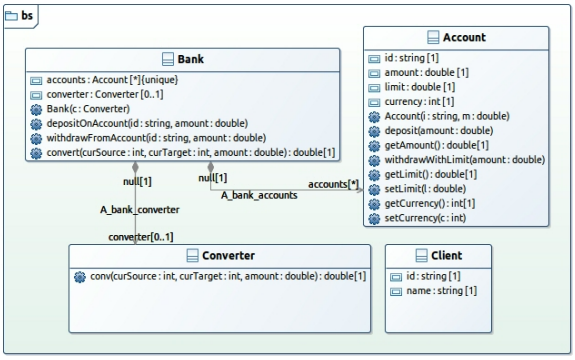
(c) BS3: {Base, Consortium}



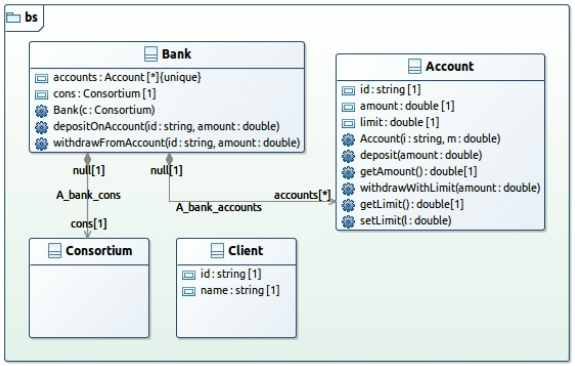
(d) BS4: {Base, Converter}



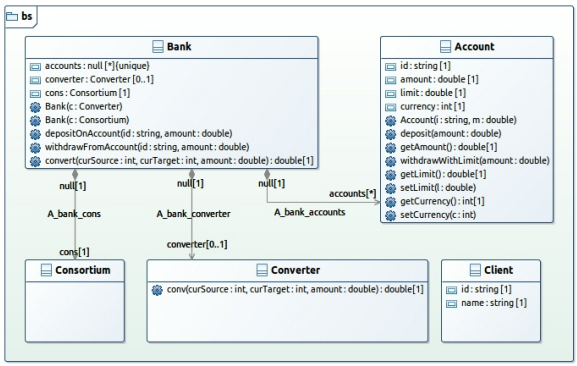
(e) BS5: {Base, Consortium, Converter}



(f) BS6: {Base, WithdrawLimit, Converter}



(g) BS7: {Base, WithdrawLimit, Consortium}



(h) BS8: {Base, WithdrawLimit, Cons., Conv.}

Figure 4.2: UML class diagram variants of BS

4.5 Experimental Settings and Implementation Details

Next we describe the experimental setup, and the implementation details of the techniques used in each step. These settings are related with the hardware used, number of runs for each technique, algorithms adopted for each step, and tools used for the implementation.

Hardware: The experiments were executed on a machine with an Intel® Core^{TM} i5-3570 CPU with 3.40GHz x 4, 16 GB of memory, and running a 64-bit Linux platform.

Independent runs: We executed 30 independent runs for each case study in the steps of FM Reengineering and Model Merging. These steps use search-based techniques, which use randomized decisions, and may reach different solutions in each run. The idea of executing many independent runs is to infer the standard behavior of these algorithms. For the steps of Feature Traceability and Variability Grafting, which use deterministic techniques, one single run was executed.

Features Traceability algorithm: For the Feature Traceability step we applied the Linsbauer et al.'s extraction algorithm [68]. This algorithm is available on the ECCO Tool[3] [46]. As described in Section 3.2, we adapted ECCO Tool to accept as input UML models.

Reverse Engineering of FMs algorithm and parameters: In our previous work we evaluated three search-based algorithms to reverse engineer FMs [9] (Appendix B). From the results we concluded that the Non-Dominated Sorting Genetic Algorithm (NSGA-II) [34] is the most promising algorithm. Based on this, in the experimentation of our approach we also used this algorithm. We used the algorithm NSGA-II of the ECJ Framework[4] [106]. The parameter settings used to configure the algorithms are shown in Table 4.2. The maximum number of fitness evaluations is the stop criteria.

Table 4.2: NSGA-II parameters

Parameter	**NSGA-II**
Number of Fitness Evaluations	200000
Population Size	200
Crossover Probability	0.7
Feature Tree Mutation Probability	0.5
CTCs Mutation Probability	0.5
Number of Elites	25%
Tournament Size	6
Maximium CTC Percentage for Builder*	0.1
Maximium CTC Percentage for Mutator*	0.5
Independent runs	30

* relative to number of features

[3] https://github.com/llinsbauer/ecco/
[4] http://cs.gmu.edu/~eclab/projects/ecj/

Model Merging algorithm and parameters: We implemented our model merging step using JMetal Framework[5] [39] and selected the mono-objective generational Genetic Algorithm (gGA) [47], similarly to our previous work [11] (Appendix C). The parameter settings used to configure gGA are presented in Table 4.3. The maximum number of fitness evaluations is the stop criteria.

Table 4.3: gGA parameters

Parameter	gGA
Number of Fitness Evaluations	4000
Population Size	200
Crossover Probability	0.95
Mutation Probability	0.2
Number of Elites	2%
Tournament Size	4
Independent runs	30

* relative to number of features

Variability Grafting algorithm: To perform the Variability Grafting step we implemented our own algorithm on top of the ECCO Tool. Our algorithm uses the trace links information discovered by the tool and enriches UML class diagrams with variabilities and features information.

4.6 Results and Analysis

This section presents the results and analysis of the experimental evaluation of the ModelVars2SPL approach. The first four subsections are devoted to the results and analysis of the solutions obtained in each step. The fifth subsection presents the discussion of the results and the answers for the research questions.

4.6.1 Features Traceability

The results of the Feature Traceability are presented in Table 4.4. For each case study we show the number of base and derivative modules found (second column), the number of dependencies in the dependency graph (third column), and the runtime (fourth column).

Regarding the module types, in Scenario 1 the case studies MMv1 and VOD are composed of only base modules, which means there are no implementations of feature interactions. On the other hand, the case studies BS and DPL have some derivative modules and ZM has many derivative modules in comparison to the number of base modules. Recalling to Table 4.1 and Figure 4.1, which presents the case studies information, we can see that ZM is one of the largest case studies and has one of the largest proportion of relationships in comparison to the other model element types. These characteristics may justify the high number of derivative modules of ZM. This also may be related to the domain characteristics that demand features to interact in order to implement the functionalities.

In Scenario 2 there are only base modules for all case studies. We infer that this happened because we selected systems that implement at least half of the features, and then

[5] http://jmetal.sourceforge.net/

Table 4.4: Feature Traceability results

Case Study	Module Type		Module Dependencies	Runtime	
	Base	Derivative		sec	msec
Scenario 1					
BS	4	3	6	1	987
DPL	7	4	10	2	206
MMv1	1	0	0	2	58
VOD	5	0	4	13	570
ZM	6	20	28	17	394
Scenario 2					
GOL	9	0	14	12	373
MMv2	2	0	1	2	230
MMv3	3	0	2	2	930
MMv4	6	0	5	3	723
MMv5	11	0	12	6	875

there are no feature interactions in the variants. Taking into account the total number of modules, MMv1 is composed of only one module. Despite of having five features (Table 4.1), the three variant models of MMv1 are similar, because the variability happens in a granularity level (statements inside operations) not represented in the UML class diagrams. MMv5 has the largest number of base modules and is the case study that has the largest number of variants.

The module dependencies represent the relationships between modules. All case studies have module dependencies, except MMv1. As expected, the number of dependencies is related to the number of modules, as we can observe analyzing the second and third columns of Table 4.4 in comparison to the fourth column. Besides, we can see that ZM, GOL and MMv5 are the case studies with the largest number of module dependencies. ZM, GOL and MMv5 are also the case studies with the largest number of relationships and operations in comparison to the other model elements (see Table 4.1 and Figure 4.1). This led us to infer that relationships and operations are related with module dependencies. These module dependencies are the basis to construct the dependency graph.

The runtime seems to be related to the number of relationships in the variants (Table 4.1). For instance, on one hand the case studies VOD, ZM, and GOL have the largest runtime and mean number of relationships in the variants, on the other hand the case study MMv5 is the largest of features and variants but smaller than the former three case studies in terms of relationships.

For a sack of illustration, Table 4.5 presents the modules of BS obtained in the Feature Traceability. The module $d^0(+Base.\{0\})$ represents the mandatory feature *Base* and it is the feature with the highest number of elements. The modules $d^1(-Converter.\{0\}, +Consortium.\{0\})$ OR $d^1(+Base.\{0\}, -WithdrawLimit.\{0\})$ represent those elements that are present when the feature *Converter* is not present and feature *Consortium* is present, or when the *Base* is present but *WithdrawLimit* is not present.

The relationships between modules are represented in a dependency graph, which is illustrated in Figure 4.3. We can observe that all modules have relationships with the module $d^0(+Base.\{0\})$, which has the mandatory feature *Base*. The number of dependencies in the graph is shown by the edge labels. This dependency graph provides information regarding the composition of the variants. For example, if we want to add the feature *WithdrawLimit* in a variant that has only feature *Base*, then we need to put seven model elements of $d^0(+WithdrawLimit.\{0\})$ together $d^0(+Base.\{0\})$ and remove two elements of the module

Table 4.5: Modules of BS

Module	Type	Elements
$d^0(+Base.\{0\})$	Base	38
$d^0(+WithdrawLimit.\{0\})$	Base	7
$d^0(+Converter.\{0\})$	Base	19
$d^0(+Consortium.\{0\})$	Base	3
$d^1(-Converter.\{0\},+Consortium.\{0\})$ OR $d^1(+Base.\{0\},-WithdrawLimit.\{0\})$	Derivative	2
$d^1(+Converter.\{0\},+Consortium.\{0\})$ OR $d^1(+WithdrawLimit.\{0\},+Consortium.\{0\})$	Derivative	1
$d^3(+Converter.\{0\},+Base.\{0\},+WithdrawLimit.\{0\},+Consortium.\{0\})$	Derivative	1

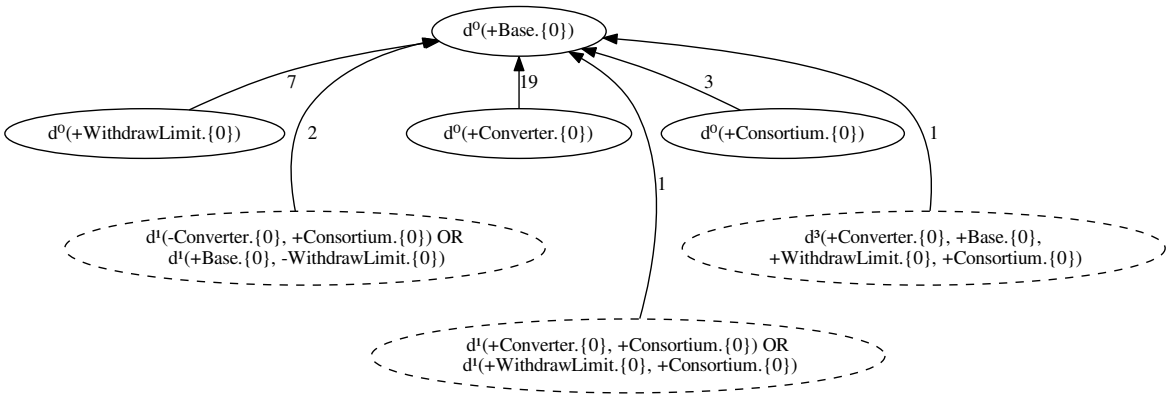


Figure 4.3: Dependency graph of BS

$d^1(+Base.\{0\},-WithdrawLimit.\{0\})$. The illustration of this task is presented in Figure 4.4 using BS variants (Figure 4.2).

4.6.2 Reverse Engineering of Feature Models

With respect to the step of Reverse Engineering of FMs, Table 4.6 presents the results of the experimental evaluation. This step uses a multi-objective search-based algorithm, thus we used the Pareto Fronts and ED indicators for the analysis. The cardinality of the Pareto Front for each case study is presented in the second column of the table. The value of the best ED and the corresponding fitness function values of the best solution are presented in the third and fourth columns, respectively. The last column of the table presents the mean runtime. The Pareto Fronts and runtime were obtained considering the 30 independent runs.

For all case studies of Scenario 1 the algorithm NSGA-II was able to reach optimal solutions (P=1.0, R=1.0, VS=1.0), leading to ED values equal to 0.0. A value equal to 1.0 for precision means that all configurations of the input variants (i.e., features set) are denoted by the reverse engineered FM; 1.0 for recall says that exactly the input products are represented, there are no surplus configurations denoted by the FM; and the best value for variability safety (VS=1.0) means that there are no violated dependencies in the FM regarding the dependency graph. These optimal solutions were possible because these case studies in Scenario 1 have variants with all the possible combinations of features. On the other hand, in Scenario 2 the best solution was reached only for two case studies, namely MMv2 and MMv3, which are the smallest case studies in this scenario (Table 4.1). For the case studies GOL, MMv4 and MMv5, the algorithm NSGA-II obtained a set of solutions, indicating a conflict among the values of the

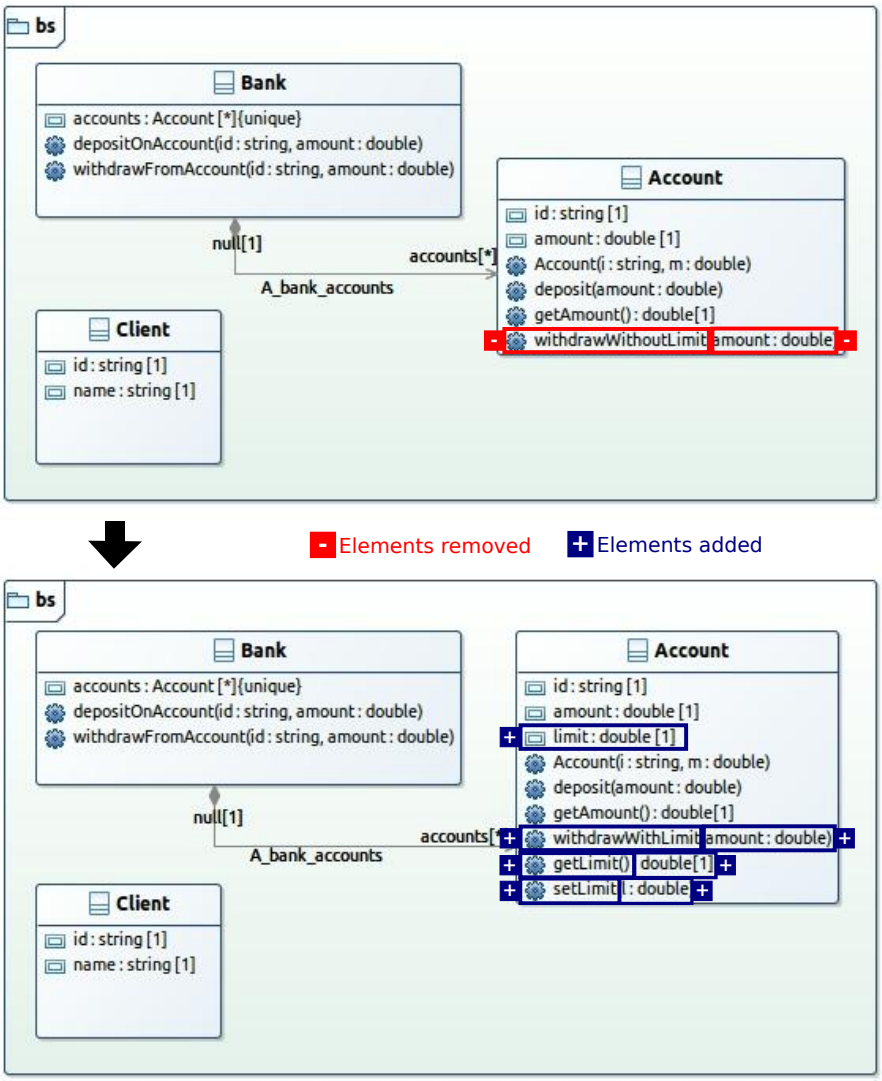


Figure 4.4: Composition of *Base* + *WithdrawLimit*, from BS1 to BS2

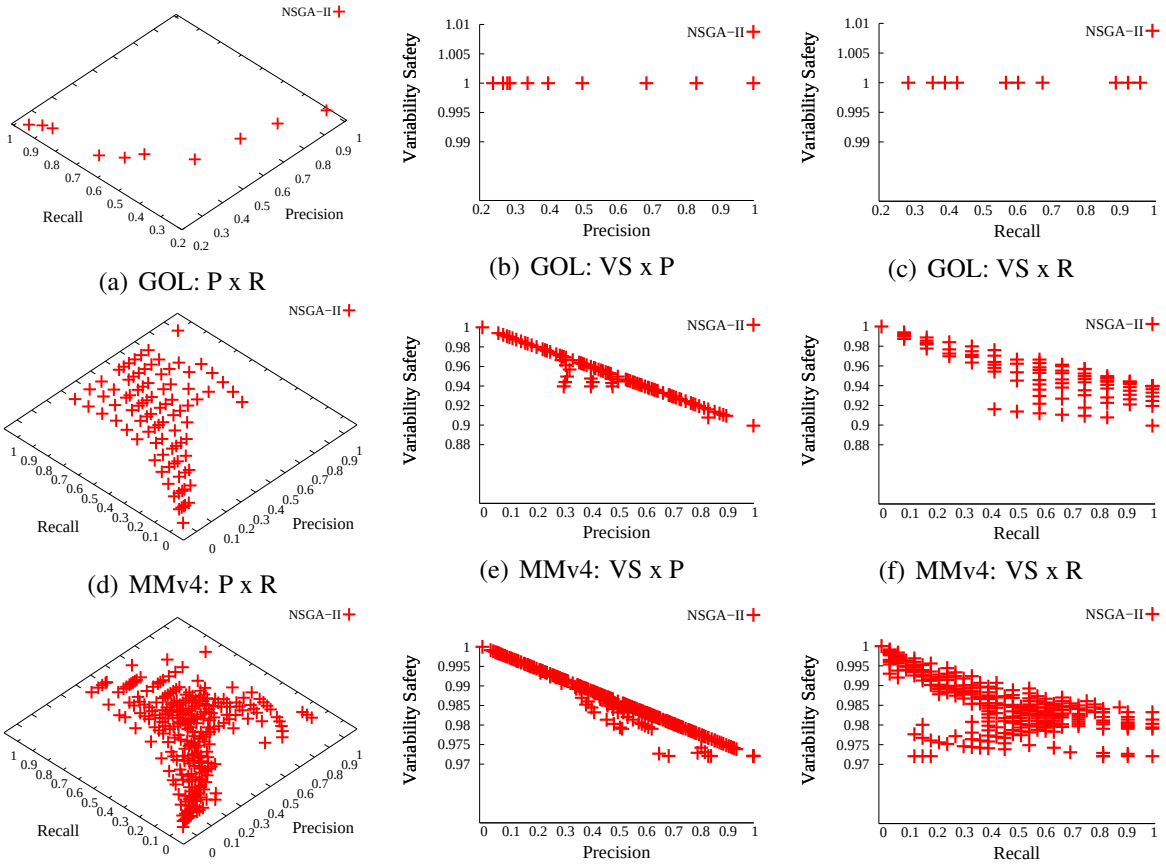
objectives being optimized. The cases with conflicting objectives take the largest amount of time. For instance, NSGA-II took almost 40 minutes to complete the evolutionary process for the case study MMv5.

Taking into account the Pareto Fronts with cardinality greater than one, Figure 4.5 presents the solutions in the search space. The graphs are presented by considering pairs of objectives, for an easy visualization. In the case study GOL, the conflicting objectives are related to P and R (Figure 4.5(a)), since the values of VS in graphs of Figures 4.5(b) and 4.5(c) are equal to 1. NSGA-II explored well the search space of GOL, as we can observe in Figure 4.5(a), the solutions cover the entire search space. Regarding case studies MMv4 and MMv5, there are interdependencies among the three objectives, but in the same way, NSGA-II explored well the search space. Based on the graphs we can see the solutions spread over the search space.

An example of FM obtained with our approach is illustrated in Figure 4.6. This FM is the optimal solution found by NSGA-II for the BS case study. Considering the eight variants of BS (Figure 4.2), the FM denotes the exact combination of features desired and does not break any dependency of the dependency graph (Figure 4.3). The tree of the FM is illustrated at the top of the figure, which has one mandatory feature *Base*, and three optional features

Table 4.6: Reverse Engineering of FMs results

Case Study	Pareto Front	Best ED	Best ED Solution			Mean Runtime		
			P	R	VS	min	sec	msec
Scenario 1								
BS	1	0.0	1.0	1.0	1.0			234
DPL	1	0.0	1.0	1.0	1.0	1	0	252
MMv1	1	0.0	1.0	1.0	1.0		5	180
VOD	1	0.0	1.0	1.0	1.0			746
ZM	1	0.0	1.0	1.0	1.0		1	395
Scenario 2								
GOL	8	0.35714	1.0	0.64286	1.0	5	37	642
MMv2	1	0.0	1.0	1.0	1.0		3	740
MMv3	1	0.0	1.0	1.0	1.0		4	820
MMv4	91	0.10063	1.0	1.0	0.89937	5	9	975
MMv5	261	0.18395	1.0	0.81818	0.97208	39	49	388



(a) GOL: P x R (b) GOL: VS x P (c) GOL: VS x R

(d) MMv4: P x R (e) MMv4: VS x P (f) MMv4: VS x R

(g) MMv5: P x R (h) MMv5: VS x P (i) MMv5: VS x R

Figure 4.5: Solutions on the search space

Converter, *Consortium*, and *WithdrawLimit*. The eight possible combinations of features (i.e., configurations) are presented at the bottom of the figure.

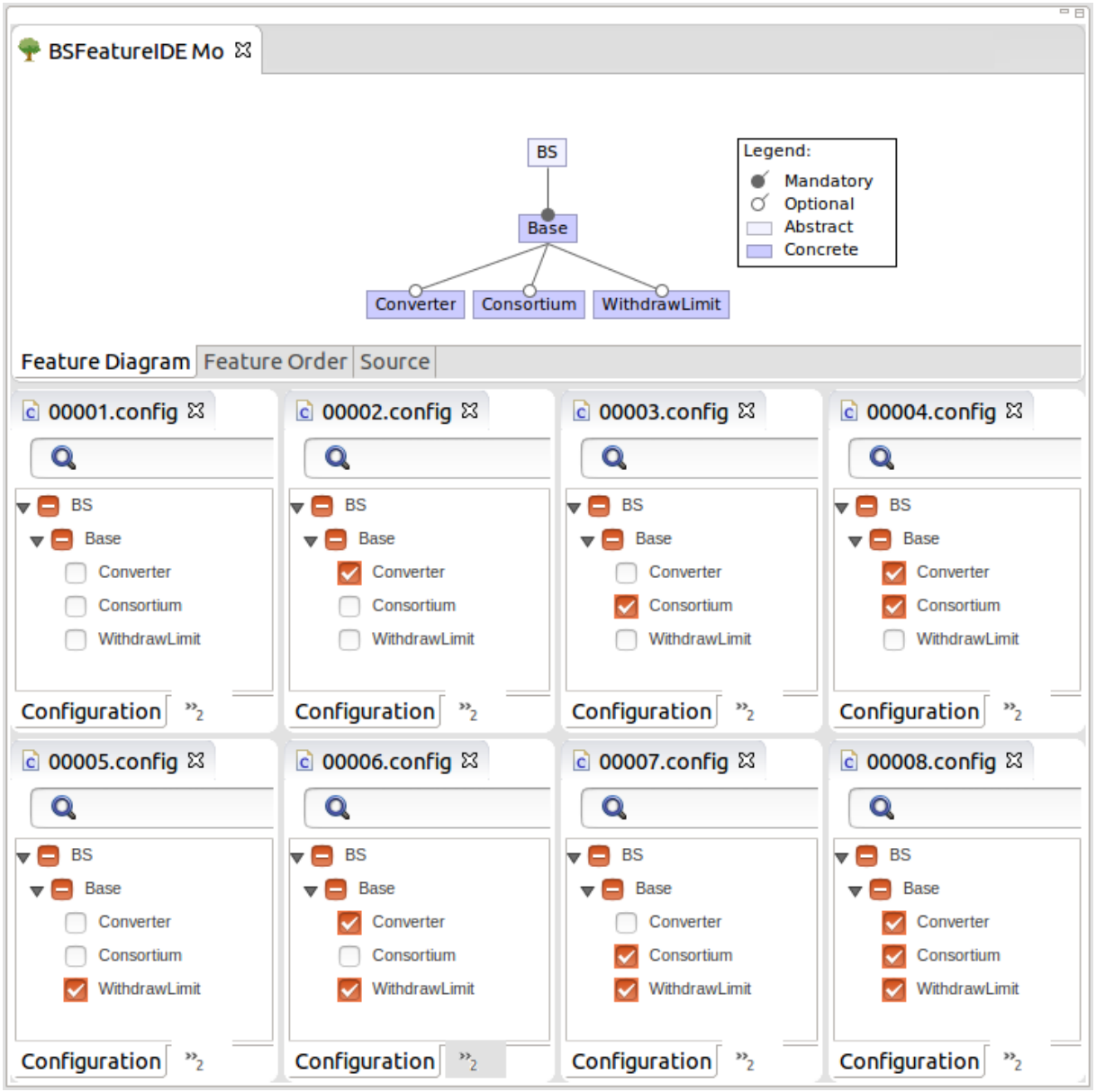


Figure 4.6: FM and configurations of BS

4.6.3 Model Merging

Our Model Merging step is based on a mono-objective optimization algorithm, thus our analysis relies on the evolution of the best solutions and the mean runtime of the 30 independent runs. Table 4.7 presents the results obtained in this step. The mean of the best solution and the best solution found considering all the independent runs are presented in the second and third columns, respectively. The numbers of model elements of the best solution are presented from the fourth to the eighth columns. The mean runtime of all runs is presented in the last column of the table.

From the fitness values of the best solutions we can observe that the most complete model obtained is not similar to any existing variant, because of the number of differences from the input variants, except for MMv1, MMv2, and MMv3 where the model variants are very similar. For instance, the merged model obtained for the case study VOD has 302 differences from the 32 input variants (Table 4.1). This means that even merging many existing differences,

Table 4.7: Model Merging results

Case Study	Mean fitness of the Best	Fitness of the Best	Elements of the Best					Mean Runtime			
			Cl	Int	Attr	Op	Rel	hour	min	sec	msec
Scenario1											
BS	20.07	20	5	0	9	15	5		1	24	724
DPL	60.53	51	5	0	21	41	5		9	6	269
MMv1	0.00	0	23	1	22	121	13		7	11	326
VOD	324.27	302	42	0	282	249	186	2	11	2	673
ZM	348.27	264	28	3	204	386	94	2	18	19	501
Scenario1											
GOL	163.63	146	19	2	33	93	39		34	31	417
MMv2	0.00	0	24	1	26	138	21		10	5	797
MMv3	0.00	0	24	1	28	141	21		20	0	752
MMv4	9.00	9	29	1	29	158	27		28	50	764
MMv5	348.27	264	36	1	46	200	30	1	23	49	51

a model completely similar to all input variants is difficult to be obtained. This situation justifies the use of search-based approaches. Analyzing the mean fitness of the best solutions we can see that the best solution is not always reached. This happens mainly because of the huge search space, and the entirely exploration of the search space is computationally infeasible. This is observed by looking at the runtimes. For two case studies, namely VOD and ZM, each independent run took more than two hours on average.

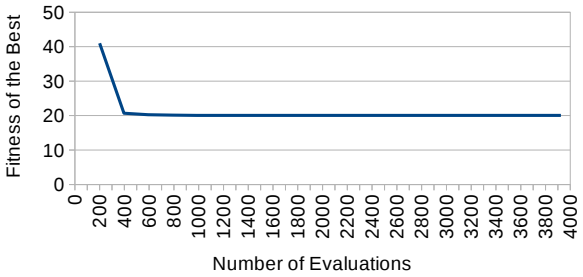
With respect to the evolution of the solutions during the evolutionary process, Figure 4.7 presents the mean values of the best solutions in each generation. In general, there is a good improvement on the fitness of the solutions in early stages of the search, and then the fitness of the best solution remains constant, except for MMv1, which already starts with an ideal solution as part of the input. From these graphs we can see that the algorithm gGA is able to explore the search space properly in the early stages of the process, but the exploitation of the search space is not possible due to computation constraints. We tried to use more computational resources during the experimentation, but then the available memory was insufficient. This happens because of the size of the individual in memory, which is an entire UML class diagram.

To illustrate the output of the Model Merging step, Figure 4.8 presents the merged model for the BS case study. This model is a complete class diagram with all the existing elements among the variants. Compared to BS8 (Figure 4.2), which is the variant with all features, this model has represented the absence of features related with the variants without the feature *WithdrawLimit*. This means that this model provides a better overview of all variants, considering both presence and absence of model elements.

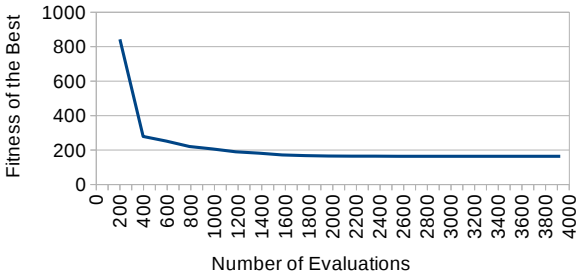
4.6.4 Variability Grafting

The results of the Variability Grafting are presented in Table 4.8. The number of model elements annotated with the traceability information is presented in the second column. The runtime for each case study is in the last column. The case studies VOD and ZM have the largest models, therefore the variability grafting algorithm took the largest runtime. However, the runtime did not take more than ten seconds.

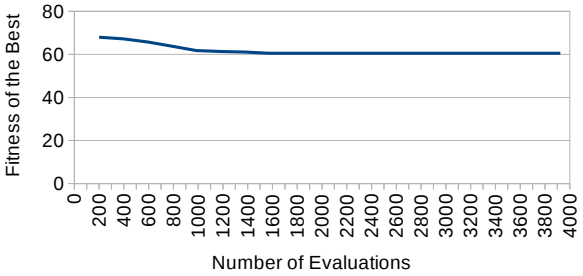
An example of PLA (i.e., an annotated model) for the BS case study is illustrated in Figure 4.9. The model is the same of Figure 4.8, except from the annotations regarding the



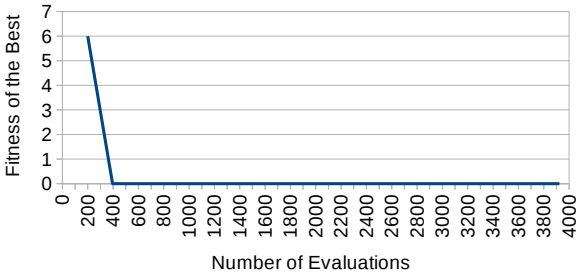
(a) BS



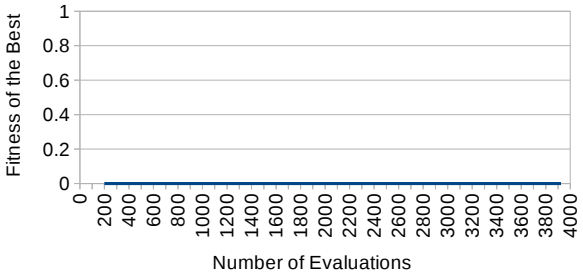
(b) GOL



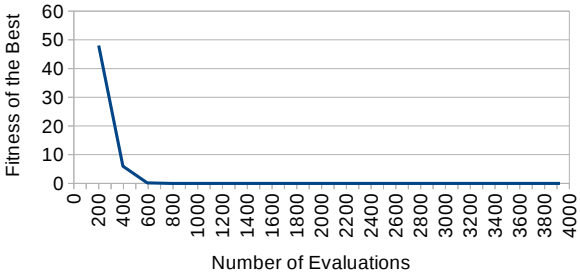
(c) DPL



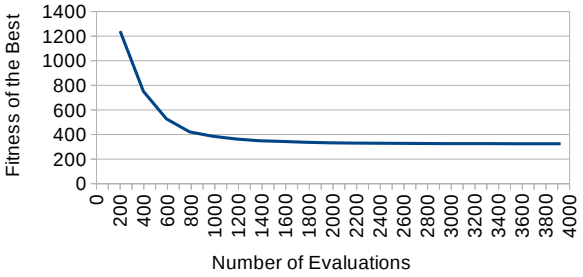
(d) MMv2



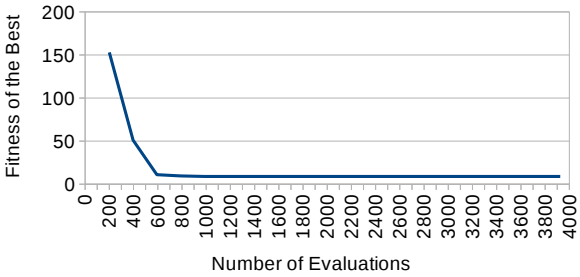
(e) MMv1



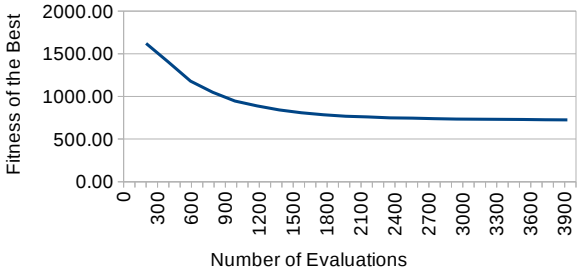
(f) MMv3



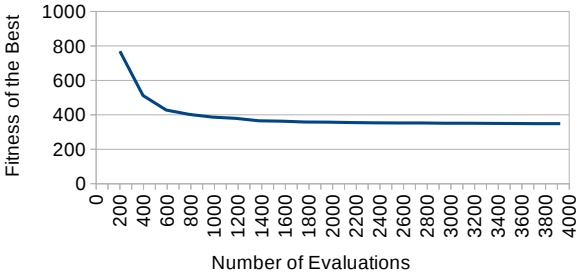
(g) VOD



(h) MMv4



(i) ZM



(j) MMv5

Figure 4.7: Evolution of the best solution

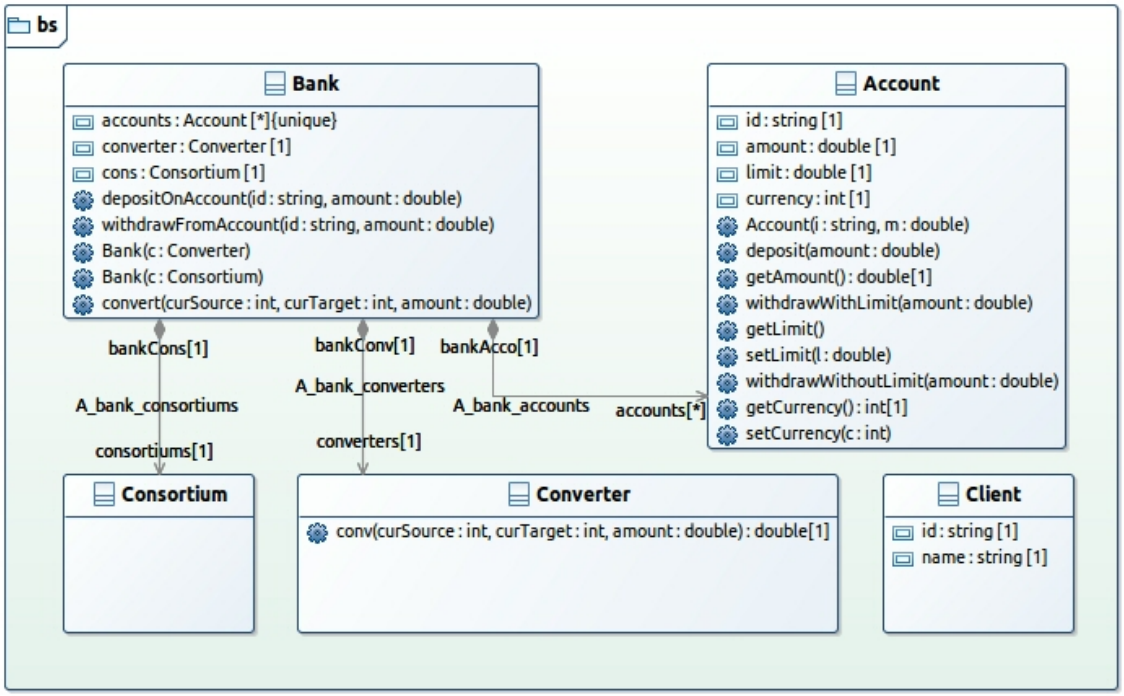


Figure 4.8: Merged UML class diagram of BS

Table 4.8: Variability Grafting results

Case Study	Model Elements Annotated	Runtime sec	Runtime msec
Scenario 1			
BS	44	1	481
DPL	103	1	811
MMv1	306	2	806
VOD	728	8	972
ZM	857	9	31
Scenario 2			
GOL	230	4	524
MMv2	327	2	899
MMv3	333	2	927
MMv4	355	3	204
MMv5	448	3	958

variabilities and features. For a sack of illustration, we present annotation of classes, attributes, and operations. The classes `Bank`, `Consortium`, and `Converter`, have their annotations depicted at the bottom of the figure. The annotated attributes `accounts`, `converter`, and `cons`, of the class `Bank` are illustrated at the top of the figure. The comment annotations for the operations `deposit()`, `withdrawLimit()`, and `withdrawWithoutLimit()` are presented on the right.

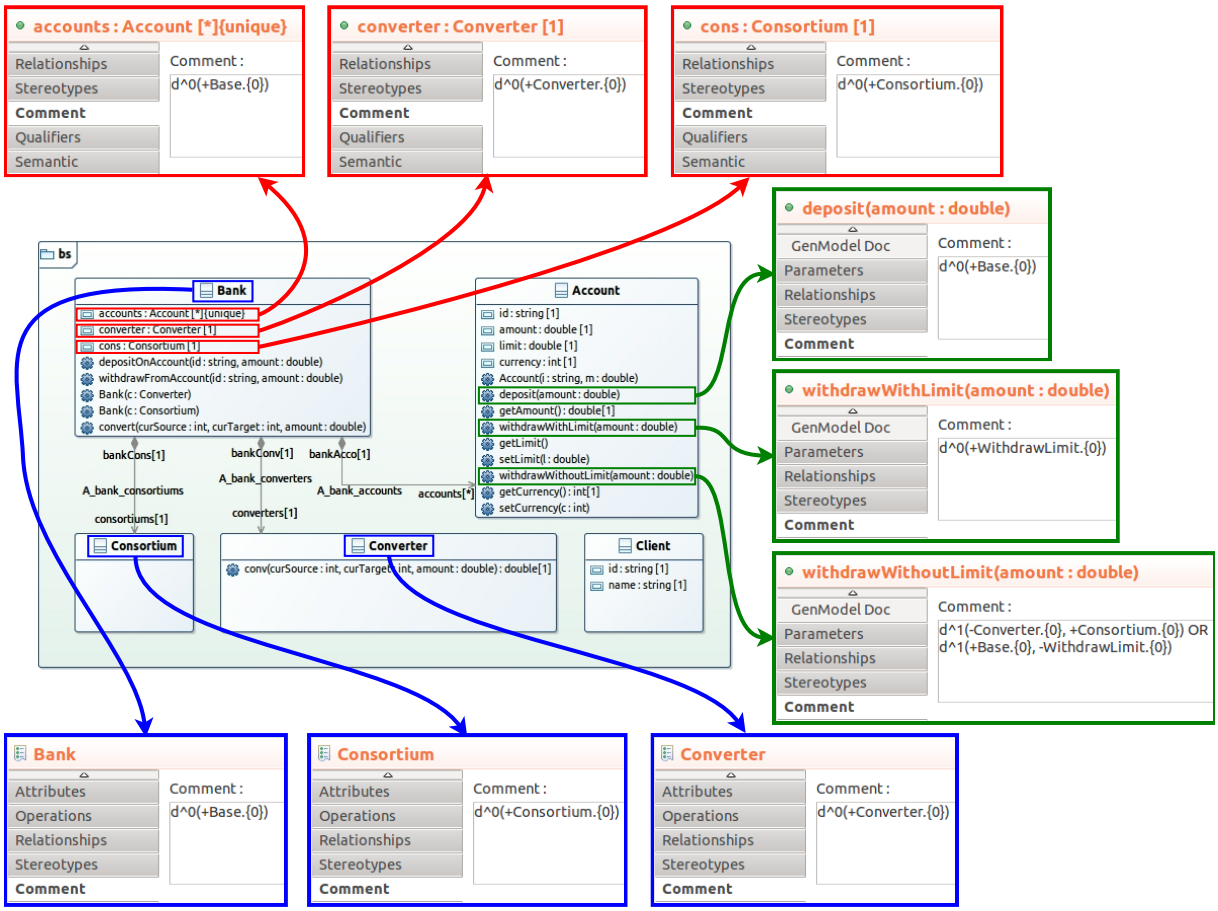


Figure 4.9: PLA of BS

4.6.5 Discussion of the Results

For the discussion of the results, we recall our research questions (Section 4.1) and provide their answers.

Answering RQ1

To answer RQ1 we summarize some results of the obtained FMs and PLAs in Table 4.9. The basis for the comparisons is the number of feature sets used as input, presented in the second column of the table; and the number of model elements of the baseline[6] UML class diagram, presented in the third column. The baseline of each case study in Scenario 1 is the UML class diagram variant that implements all its features. For Scenario 2 the baseline of each case study is the most complete UML class diagram variant known (i.e., that implements the greater number of features) but were not included as input. Regarding the outputs of ModelVars2SPL, for the obtained FM we present in the table the number of contained feature sets (fourth column), the number of missing feature sets (fifth column), and the number of violated dependencies (sixth column) in comparison to the input feature sets. For the obtained PLAs we present the number of model elements (seventh column), the number of differences from the baseline to the PLA (eighth column), and the number of differences from the PLA to the baseline (ninth column).

To evaluate the quality of the ModelVars2SPL solutions, the first point to be taken into account is the instantiation of the same input variants as products of the SPL. Initially we consider

[6]The baseline variants are indicated in Appendix D.

Table 4.9: Results for the obtained FMs and PLAs

Case Study	Input Feature Sets	Baseline Model Elements*	Obtained FM			Obtained PLA		
			Contained Feature Sets	Missing Feature Sets	Violated Dependencies	Model Elements*	Differences Baseline>PLA	Differences PLA>Baseline
Scenario1								
BS	8	32	8	0	0	34	16	3
DPL	16	72	16	0	0	72	303	2
MMv1	3	180	3	0	0	180	827	0
VOD	32	675	32	0	0	759	2122	36
ZM	32	656	32	0	0	715	2588	99
Scenario2								
GOL	28	178	18	10	0	186	1954	521
MMv2	4	210	4	0	0	210	953	0
MMv3	9	214	9	0	0	215	1989	712
MMv4	12	245	12	0	12	244	2195	819
MMv5	33	313	27	6	27	313	2788	1038

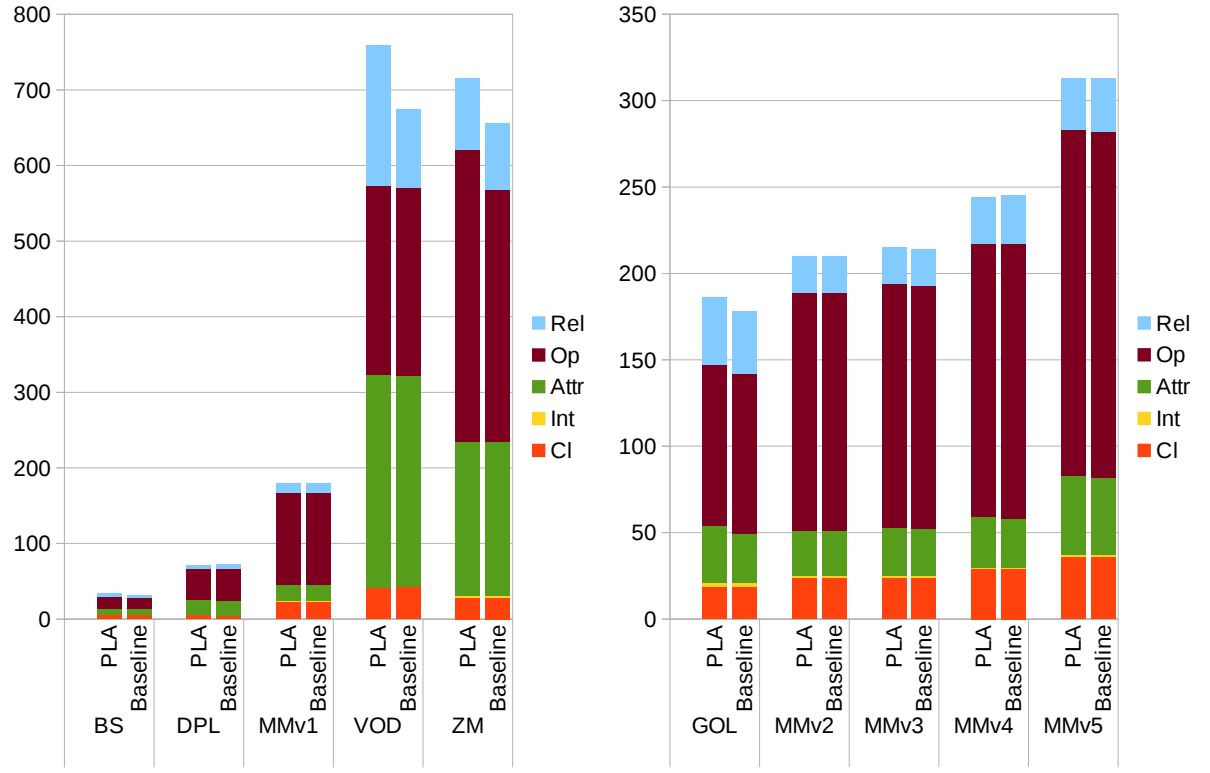
*(Classes + Interfaces + Attributes + Operations + Relationships)

the obtained FM for each case study. Observing the feature sets used as input (second column) and the contained feature sets in the obtained FMs (fourth column) presented in Table 4.9, we can see that our approach is able to represent exactly the same input configurations of products for eight out of ten case studies. Some feature sets used as input are missing only in the case studies GOL and MMv5. Taking into account the situations where there exists a set of input variants that well represent combinations of features (e.g., Scenario 1) our approach can reach optimal FMs. In situations where the variants implement only few features (e.g., Scenario 2), our approach reached optimal solutions for three out of five case studies. For Scenario 2 it was expected that the reverse engineering of FM would be a difficult task, because these case studies has only variants that implement at most half of the features. Despite of not to find optimal solutions for two case studies, the obtained FMs have more contained feature sets than missing ones (fifth column). For instance, in the GOL case study the obtained FM denotes 18 input feature sets, and only ten feature sets are missing. With respect to the violated dependencies, only in two case studies the obtained FM denotes feature sets that do not satisfy some dependencies of the dependency graph, namely MMv4 and MMv5.

Regarding the PLA, we analyze the baseline in comparison to the PLA obtained with our approach. In Table 4.9 we can observe that the number of model elements in the PLA (seventh column) is greater than the model elements in the baseline (third column), except for MMv4. The greater number of model elements indicates that the PLA is more complete than the baseline. To corroborate our findings, Figure 4.10 presents the graphs with the number of model elements; namely classes (Cl), interfaces (Int), attributes (Attr), operations (Op), and relationships (Rel); for the baseline variant and the PLA of each case study.

In Figure 4.10(a) we can observe that the number of model elements in the PLA is very similar to the baseline in the case studies BS, DPL and MMV1, the smallest case studies of Scenario 1 (see Figure 4.1). For the case studies VOD and ZM, the PLA has more model elements than the baseline. This happens because these case studies are large systems, and they have many model elements that implement the functionalities. In Figure 4.10(b) we can see that both the PLA and the baseline are very similar in terms of model elements. There is a small difference in the case study GOL, where the PLA has more model elements than the baseline, which is expected.

In the last two columns of Table 4.9 we can see the number of differences between the baseline and PLA taking into account the two directions of comparison. For instance, considering the BS case study, the baseline has 16 different model elements in comparison to the PLA. On the other hand, the PLA has only 3 differences in comparison to the baseline. In the analysis of



(a) Scenario 1 (b) Scenario 2

Figure 4.10: Baselines and PLAs

these differences in all case studies we can clearly see that the PLA is much more similar to the baseline than the other way around.

RQ1 Discussion: Based on the results presented above, we can evaluate the quality of obtained solutions in terms of correctness (see Section 4.2). We can attest that the FMs and PLAs generated with ModelVars2SPL have high *accuracy*, since they well represent the input variants. The FM most of the times represents exactly the desired input feature sets, and the PLA is very similar to the baselines, indicating that there is no *redundancy* and having a very good *completeness*. Analyzing that both FMs and PLAs well represent the input variants, we can affirm that the *reliability* is very good. Regarding the FMs generated considering constraints in feature combinations, we can attest that the *robustness* is also very satisfactory. In summary, we can say that the correctness of the solution obtained with ModelVars2SPL is very high.

Answering RQ2

To answer RQ2 we have to consider the solutions of all steps. In summary, the step Features Traceability obtained solutions as expected, discovering properly the base modules and derivative modules, and dependencies (Table 4.4); the reverse engineered FMs in the second step well describe the input feature sets, the majority of the solutions found with the search-based technique were optimal solutions in the Pareto front with the best values of ED (Table 4.6); the evolutionary process in the step of Model Merging was able to reach good global UML class diagrams by exploring the search space, as previously discussed, the solutions are very similar to the baseline variants (Table 4.7); and we observed that our algorithm in the Variability Grafting step can successfully include annotations in model elements to generate good PLAs (Table 4.8).

Another discussion is about the runtime to obtain a solution using ModelVars2SPL. In the previous section we presented the runtime of each step individually, but here we consider it entirely. Table 4.10 presents the runtime of the four steps of the approach and the total runtime. To obtain the FM and PLA for the BS case study the approach took only 1 minute and 28 seconds, this was the fastest case study executed. On the other hand, for the case studies VOD, ZM, and MMv5 the approach took more than 2 hours to be finished. These case studies have the largest number of variants and model elements (see Table 4.1). The times are very short if we consider the manual effort of a human expert to perform the same task. Furthermore, the algorithms were executed in a desktop computer.

Table 4.10: Approach runtime

Case Study	Feature Traceability		Reverse Engineer of FMs			Model Merging				Variability Grafting		Total Runtime			
	sec	msec	min	sec	msec	hour	min	sec	msec	sec	msec	hour	min	sec	msec
Scenario 1															
BS	1	987			234		1	24	724	1	481		1	28	426
DPL	2	206	1	0	252		9	6	269	1	811		10	10	538
MMv1	2	58		5	180		7	11	326	2	806		7	21	370
VOD	13	570			746	2	11	2	673	8	972	2	11	25	962
ZM	17	394		1	395	2	18	19	501	9	31	2	18	47	322
Scenario 2															
GOL	12	373	5	37	642		34	31	417	4	524		40	25	956
MMv2	2	230		3	740		10	5	797	2	899		10	14	667
MMv3	2	930		4	820		20	0	752	2	927		20	11	429
MMv4	3	723	5	9	975		28	50	764	3	204		34	7	667
MMv5	6	875	39	49	388	1	23	49	51	3	958	2	3	49	272

RQ2 Discussion: Considering the artifacts obtained in each step we can attest that ModelVars2SPL can successfully automate the specific phases of the reengineering process, as well as the entire process. The high quality of the FMs and the PLAs, discussed in the answer of RQ1, is result of good solutions obtained in each distinct step. Furthermore, the runtime of the steps allows the approach being applied in practice, quickly providing good SPL solutions for programmers, engineers and architects to follow the entire process of reengineering.

4.7 Example of Use of the ModelVars2SPL Solutions

An example on how to use the solutions obtained with ModelVars2SPL for the BS case study is illustrated in Figure 4.11. The FM and the PLA can be used together to see which elements are involved in the implementation of features, easing the understanding of the products variants. For instance, in the figure we highlighted all model elements related to the implementation of the feature *Base*, which is mandatory in the FM, and are annotated with the comment $d^0(+Base.\{0\})$. Recalling the guidelines for using ModelVars2SPL solutions (Section 3.6), in case of a requirement for including an attribute `birth_date` in the class `Client`, as part of the feature *Base*, this can be done once in the PLA, and then it will be present in all products of the BS system. In case of the initial variants constructed using the C&O approach, this maintenance task should be done eight times, one time for each variant.

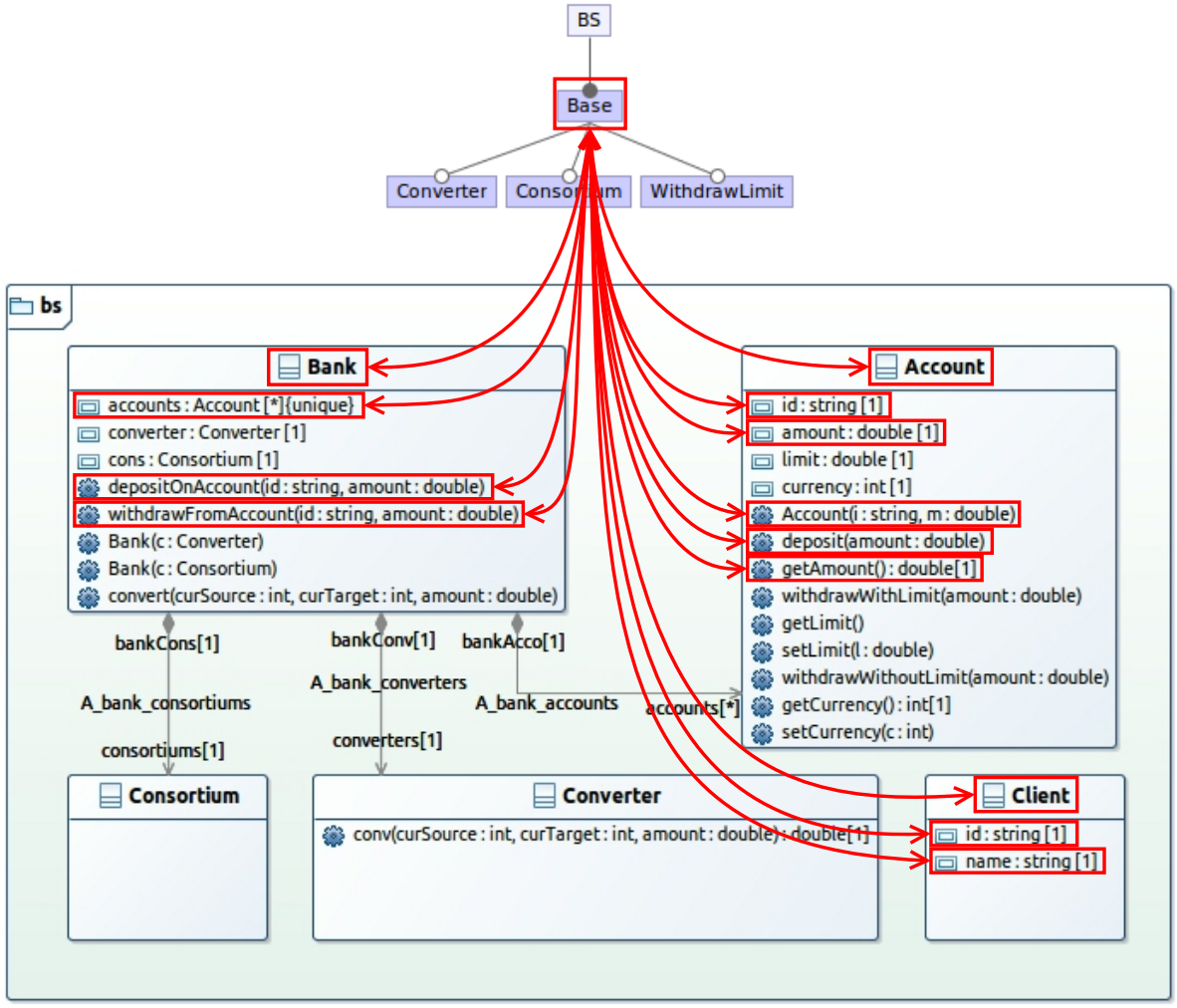


Figure 4.11: Example of traceability between the FM and the PLA for BS

4.8 Threats to Validity

The first threat to validity identified was the parameter settings for the search-based algorithms. We addressed this threat by using parameter values recommended in the literature [39, 106].

The second threat is the influence of the case studies. Despite of using only ten case studies, these systems are from different domains, and have different sizes in terms of variants, features, and model elements. We used two scenarios to have a better experimental evaluation of our approach. Based on that, we think these case studies can provide evidence about the soundness of our approach. But nonetheless further studies should be conducted in the future.

The third threat concerns the comparison to other approaches. As baseline we used models known in advance that implement all features that compose the systems, assuming that these models are the closest to an ideal solution.

The fourth threat is related to the set of measures we used to evaluate the solutions. There is a lack of measures designed to evaluate reuse of reengineered systems [10]. To reduce this threat we used the same measures used in similar works [102, 105].

The fifth threat to validity is the scalability of our approach. We did not perform an in-depth analysis regarding the behavior of our approach controlling the size of the input to see

until which point the solutions remain good. However, we tried to mitigate this threat by using case studies with different sizes in terms of number of variants, features, and model elements.

The last threat refers to the validation of the solution by practitioners. Certainly, because there are no canonical representations of FMs and PLAs, domain knowledge can play a significant role when choosing among different solutions.

4.9 Concluding Remarks

In this chapter we presented the details of the experimentation used to evaluate the solutions obtained with ModelVars2SPL. The goal of the evaluation was to answer two research questions regarding (i) the correctness of the FM and PLA obtained, and (ii) the output of each step of the approach. In the experiments we used ten case studies in two scenarios, different in terms of number of different features implemented in each variant. The analysis of the results allowed us to answer the two research questions as follows: the FMs and PLAs obtained with ModelVars2SPL have a high degree of correctness, and the output of each step is good in accordance to the goal of the phases of the reengineering process.

Chapter 5

Conclusion

This dissertation presents an approach to reengineer model variants into SPLs. The ModelVars2SPL approach is composed of four steps, namely Features Traceability, Reverse Engineering of FMs, Model Merging, and Variability Grafting, covering the entire reengineering process. The input for the approach is a set of UML class diagram variants and the set of features that each variant implements. The output is an FM, which describes the configuration of the variants; and a PLA, which represents the model elements related to each feature. To deal with the complex tasks of reverse engineering of FM and model merging, our approach uses search-based techniques.

Besides the benefits of using a systematic reuse approach, our approach exploits the use of UML design models. The use of UML models can support the reengineering process in the design level, being independent of programming language. In addition, programmers, engineers, and architects can have a broader view of the SPL.

We designed and conducted an experiment to evaluate the quality of the results obtained with the proposed approach. The quality of the SPL artifacts (i.e., FMs and PLAs) was measured by considering how well they represent the input variants. Furthermore, we also evaluated the quality of each intermediary output generated in each step of the approach taking into account the goal of each phase of the reengineering process. For the experimentation we used ten case studies in two scenarios. The first scenario is composed of variants with all feature combinations known, and the second scenario with only half of the available configurations known. For the analysis we considered the runtime, cardinality of the Pareto Front, Euclidean distance from ideal solutions, and evolution of the best solutions along the evolutionary process.

The results indicate that ModelVars2SPL is effective to obtain FMs and PLAs with high degree of correctness compared to the input variants. Our approach reached FMs that represent the exact set of features sets used as input, and the constructed PLAs are similar to the baselines, however, more complete than existing variants. Furthermore, the output of each step corresponds to the expected solutions for the reengineering phases. To perform all steps of the approach, in the worst situation, the runtime was less than 2 hours and 30 minutes.

With respect to the goals of this dissertation, we implemented a fully automated approach to reengineer UML models into an SPL. There is no need of human experts and the approach produces an output for each step, allowing practitioners follow the reengineering process. We believe that ModelVars2SPL might motivate and support the adoption of systematic reuse (i.e., SPLs) in industry. In addition to this, in the next section we describe a complete list of our contributions in this dissertation.

5.1 Contributions

The contributions of this doctoral work are:

- *Definition of the reengineering process.* As a result of our systematic mapping study (Appendix A) we proposed a process of reengineering existing variants into SPLs. We have not found any definition like that so far. This definition can support new researches to propose new tools or approaches and ease the comparison among different approaches for specific phases of the reengineering process.
- *Enabling the traceability of UML models.* The first step of the reengineering process is to obtain the traceability between features and implementation artifacts. For this purpose we have decided to extend the tool ECCO. Regarding the focus of our work, we extended the functionalities of the tool to work with EMF UML models as input.
- *Reverse engineering of FMs considering dependencies among artifacts.* The second step of our approach is devoted to reverse engineer a variability model from model variants. In our initial work of reverse engineering of feature models considering dependencies among artifacts we used source-code dependencies [8, 9] (Appendix B). In this doctoral work we reverse engineer feature models by using dependencies extracted from UML model elements.
- *Model merging of multiple UML model variants.* From the best of our knowledge, the search-based merging approaches existing in the literature work with at most two or three models at once. The third step of our approach performs the merging of multiple models. For instance, for a set of sixteen product variants our approach is able to merge the models of all these variants.
- *State-based model merging using a search-based technique.* Our merging strategy relies on a search-based technique. Existing studies on merging models using search-based techniques are history-based approaches, which means they work on sequence of changes used to transform one model in another model. Our approach works with the model itself, it is not required to know the changes in the models.
- *Variability grafting in UML models.* In the context of our work, an architecture of an SPL is a design model defining the static architecture of the family of systems through a maximal UML class diagram and feature-related annotations. To obtain this artifact we graft the variability in a UML model.

5.2 Research Limitations

During the conduction of this doctoral work we identified some limitations. These limitations are described next:

- The first limitation is related to the input of ModelVars2SPL. We use UML class diagrams as input, what can limit the wide use of our approach. One way to make the approach suitable for different scenarios is to deal with metamodels that represents a wide set of models, such as Meta Object Facility (MOF)[1]. This would also collaborate to model-driven development of SPLs.

[1] `http://www.omg.org/spec/MOF/`

- The second limitation concerns about the output of ModelVars2SPL. The SPLE encompasses the development of infrastructures, models, technical activities, definition of roles, artifacts, source code, etc. Currently, our approach is restricted to FMs and PLAs. Despite we do not cover the entire SPLE life cycle, we provide a starting point to extract an SPL from model variants.

- Another limitation known is the lack of evaluation of the proposed approach scalability. Despite of ModelVars2SPL effectively dealing with the case studies used in the experimentation, we have not evaluated until which point it would be still working. We can argue that for large input set of variants, or for very large models, it is possible to fragment the input in some subsets and then use these different subsets as input. Finally, the solutions of the subsets could be used as input for a new execution of ModelVars2SPL.

- Along of this dissertation we present some advantages of having a fully automated approach, such as reducing the human effort. However, we can point as limitation the drawback of reaching solution without consider human desire. For example, there are situations were professionals prefer keep semantics in the models instead of having an improved structure of the model. Our approach does not consider such situation, but a solution reached by our approach can be modified manually to fit professional desires.

- Another limitation is to deal with the semantics of the models. It was out of our scope to consider the semantics during the reengineering process. However, the semantics can be a factor to be included in the reengineering process with the goal of generating semantically well formed models.

5.3 Future Research Directions

We identified some future directions that might be the focus of further research. Among them, we can mention:

- *Automatic product architecture instantiation*: our approach is able to obtain a PLA from a set of model variants, which supports the analysis, development, maintenance, and evolution of SPLs; however, another goal of a PLA is to support the instantiation of product architectures of the SPL. A research opportunity is the development of tools to automatically instantiate products from PLAs based on UML class diagrams with features annotated.

- *Reconciliation of designs from different stakeholders*: our work is based on model variants created using ad hoc strategies for software reuse. Nonetheless, models are commonly used as communication language between stakeholders during the design of complex systems. Based on the idea that different stakeholders might create models with similarities and specificities, the use of our approach to reconcile such models would be very beneficial.

- *Refactoring recommendations for PLAs*: the merging of many model variants can generate one model that might have bad smells, for instance with a god class. Based on this, studies on refactoring recommendations would ease the task of improving the quality of SPL designs.

- *Metrics to evaluate reuse opportunities*: along the development of this doctoral work we faced the lack of reuse-based metrics. For instance, what are the characteristics of existing

system variants that lead to the necessity of SPL migration? After what point, regarding the number of variants or features, is it worth the reengineering of systems into SPLs?

- *Experimentation with more case studies and scenarios*: another research direction is the evaluation of our approach using more case studies, for example using systems with a greater number of features. This would allow evaluating scalability of the steps of our approach. Different scenarios such as systems in the same domain may be focus for new studies.

- *Further analysis on the applications of the solutions*: a future research opportunity is the investigation of the use of reengineered SPLs to support activities of bug fixing, detection of bad smells and refactoring, evolution of the SPL with new products, improvement of the performance of the products, optimization of the variability management, and better design of test cases.

- *Integration of ModelVars2SPL with approaches of architecture improvement*: some works on the field of SPL are devoted to optimize existing PLAs [28]. Future work may be the use of the ModelVars2SPL outputs as input to approaches of PLA optimization.

Bibliography

[1] E. Abbasi, M. Acher, P. Heymans, and A. Cleve. Reverse engineering web configurators. In *IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE)*, pages 264–273, 2014.

[2] M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire. On extracting feature models from product descriptions. In *6th International Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS, pages 45–54, New York, NY, USA, 2012. ACM.

[3] R. Al-msie'deen, A.-D. Seriai, M. Huchard, C. Urtado, and S. Vauttier. Mining features from the object-oriented source code of software variants by combining lexical and structural similarity. In *IEEE 14th International Conference on Information Reuse and Integration (IRI)*, pages 586–593, Aug 2013.

[4] R. AL-Msie'deen, A. D. Seriai, M. Huchard, C. Urtado, S. Vauttier, and H. Salman. An approach to recover feature models from object-oriented source code. *Journée Lignes de Produits*, pages 15–26, 2012.

[5] N. Ali, W. Wu, G. Antoniol, M. Di Penta, Y. Gueheneuc, and J. Hayes. Moms: Multi-objective miniaturization of software. In *27th IEEE International Conference on Software Maintenance (ICSM)*, pages 153–162, Sept 2011.

[6] E. Almeida, J. Mascena, A. Cavalcanti, A. Alvaro, V. Garcia, S. Lemos Meira, and D. Lucrédio. The domain analysis concept revisited: A practical approach. In M. Morisio, editor, *Reuse of Off-the-Shelf Components*, volume 4039 of *Lecture Notes in Computer Science*, pages 43–57. Springer Berlin Heidelberg, 2006.

[7] S. Apel, D. Batory, C. Kästner, and G. Saake. *Feature-Oriented Software Product Lines*. Springer, 2013.

[8] W. K. G. Assunção, R. E. Lopez-Herrejon, L. Linsbauer, S. R. Vergilio, and A. Egyed. Extracting variability-safe feature models from source code dependencies in system variants. In *Genetic and Evolutionary Computation Conference (GECCO)*, pages 1303–1310. ACM, 2015.

[9] W. K. G. Assunção, R. E. Lopez-Herrejon, L. Linsbauer, S. R. Vergilio, and A. Egyed. Multi-objective reverse engineering of variability-safe feature models based on code dependencies of system variants. *Empirical Software Engineering*, pages 1–32, 2016.

[10] W. K. G. Assunção, R. E. Lopez-Herrejon, L. Linsbauer, S. R. Vergilio, and A. Egyed. Reengineering legacy applications into software product lines: A systematic mapping. *Empirical Software Engineering*, pages 1–45, 2017.

[11] W. K. G. Assunção, R. E. Lopez-Herrejon, and S. R. Vergilio. Discovering software architectures with search-based merge of UML model variants. In *16th International Conference on Software Reuse (ICSR 2017)*, pages 1–16, 2017. To appear.

[12] W. K. G. Assunção and S. R. Vergilio. Feature location for software product line migration: A mapping study. In *2nd International workshop on Reverse Variability Engineering (REVE 2014)*, pages 1–8, 2014.

[13] F. Bachmann and P. Clements. Variability in software product lines. Technical Report CMU/SEI-2005-TR-012, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2005.

[14] E. Bagheri, F. Ensan, and D. Gasevic. Decision support for the software product line domain engineering lifecycle. *International Conference on Automated Software Engineering*, 19(3):335–377, Sept. 2012.

[15] D. Batory. Product-line architectures. In *Smalltalk and Java Conference*, 1998.

[16] J. Bayer, T. Forster, D. Ganesan, J.-F. Girard, I. John, J. Knodel, R. Kolb, and D. Muthig. Definition of reference architectures based on existing systems. Technical Report Report No. 034.04/E, Fraunhofer IESE-Report No. 034.04/E, 2004.

[17] G. Bécan, M. Acher, B. Baudry, and S. Ben Nasr. Breathing ontological knowledge into feature model management. Technical Report RT-0441, INRIA - Institut National des Sciences Appliquées (INSA) - Rennes - Université de Rennes 1, Oct 2013.

[18] D. Benavides, S. Segura, and A. R. Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615–636, 2010.

[19] D. Benavides, S. Segura, P. Trinidad, and A. R. Cortés. FaMa: Tooling a framework for the automated analysis of feature models. In *International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*, pages 129–134, 2007.

[20] D. Berardi, D. Calvanese, and G. D. Giacomo. Reasoning on UML class diagrams. *Artificial Intelligence*, 168(1):70 – 118, 2005.

[21] T. Berger, S. She, R. Lotufo, A. Wasowski, and K. Czarnecki. A study of variability models and languages in the systems software domain. *IEEE Transactions on Software Engineering*, 39(12):1611–1640, 2013.

[22] R. Capilla, J. Bosch, K.-C. Kang, et al. *Systems and software variability management: Concepts Tools and Experiences*. Springer, 2013.

[23] L. Chen and M. A. Babar. Variability management in software product lines: An investigation of contemporary industrial challenges. In *14th International Conference Software Product Lines (SPLC)*, pages 166–180, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[24] E. Chikofsky and I. Cross, J.H. Reverse engineering and design recovery: a taxonomy. *IEEE Software*, 7(1):13–17, 1990.

[25] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[26] R. Cloutier, G. Muller, D. Verma, R. Nilchiani, E. Hole, and M. Bone. The concept of reference architectures. *Systems Engineering*, 13(1):14–27, 2010.

[27] J. Cochrane and M. Zeleny. *Multiple Criteria Decision Making*. University of South Carolina Press, Columbia, 1973.

[28] T. E. Colanzi and S. R. Vergilio. Applying search based optimization to software product line architectures: Lessons learned. In G. Fraser and J. Teixeira de Souza, editors, *Search Based Software Engineering*, volume 7515 of *Lecture Notes in Computer Science*, pages 259–266. Springer Berlin Heidelberg, 2012.

[29] T. E. Colanzi, S. R. Vergilio, W. K. G. Assunção, and A. Pozo. Search based software engineering: Review and analysis of the field in brazil. *Journal of Systems and Software*, 86(4):970–984, 2013. SI : Software Engineering in Brazil: Retrospective and Prospective Views.

[30] A. C. J. Contieri, G. G. Correia, T. E. Colanzi, I. M. S. Gimenes, E. A. Oliveira Junior, S. Ferrari, P. C. Masiero, and A. F. Garcia. Extending UML components to develop software product-line architectures: Lessons learned. In I. Crnkovic, V. Gruhn, and M. Book, editors, *Software Architecture*, volume 6903 of *LNCS*, pages 130–138. Springer Berlin Heidelberg, 2011.

[31] B. Coulange. *Software reuse*. Springer Science & Business Media, 2012.

[32] R. Damaševičius, P. Paškevičius, E. Karčiauskas, and R. Marcinkevičius. Automatic extraction of features and generation of feature models from Java programs. *Information Technology And Control*, 41(4):376–384, 2012.

[33] A. L. de Oliveira, F. C. Ferrari, R. T. Braga, R. A. Penteado, and V. V. de Camargo. Restructuring frameworks towards framework product lines. In *Latin American Workshop on Aspect-Oriented Software Development (LA-WASP'12)*, pages 1–2, 2012.

[34] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6(2):182 –197, Apr. 2002.

[35] S. Demeyer, S. Ducasse, and O. Nierstrasz. *Object-oriented reengineering patterns*. Square Bracket associates, Switzerland, 2009. Version of 2009-09-28.

[36] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk. Feature location in source code: a taxonomy and survey. *Journal of Software: Evolution and Process*, 25(1):53–95, 2013.

[37] L. Dobrica and E. Niemela. A survey on software architecture analysis methods. *IEEE Transactions on Software Engineering*, 28(7):638–653, 2002.

[38] Y. Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker, and K. Czarnecki. An exploratory study of cloning in industrial software product lines. In *17th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 25–34, March 2013.

[39] J. J. Durillo and A. J. Nebro. jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software*, 42:760–771, 2011.

[40] S. Duszynski, J. Knodel, and M. Becker. Analyzing the source code of multiple software variants for reuse potential. In *18th Working Conference on Reverse Engineering (WCRE)*, pages 303–307, Oct 2011.

[41] H. Eyal Salman, A. Djamel Seriai, C. Dony, and R. Al-Msie'Deen. Identifying traceability links between product variants and their features. In *1st International workshop on Reverse Variability Engineering (REVE)*, pages 17–22, Genova, Italie, Mar 2013.

[42] H. Eyal-Salman, A.-D. Seriai, C. Dony, and R. Al-msie'deen. Recovering traceability links between feature models and source code of product variants. In *VARiability for You Workshop: Variability Modeling Made Useful for Everyone*, VARY 2012, pages 21–25, New York, NY, USA, 2012. ACM.

[43] D. Faust and C. Verhoef. Software product line migration and deployment. *Software: Practice and Experience*, 33(10):933–955, 2003.

[44] A. Ferrari, G. O. Spagnolo, and F. Dell'Orletta. Mining commonalities and variabilities from natural language documents. In *17th International Software Product Line Conference*, SPLC'13, pages 116–120, New York, NY, USA, 2013. ACM.

[45] E. Figueiredo, N. Cacho, C. Sant'Anna, M. Monteiro, U. Kulesza, A. Garcia, S. Soares, F. Ferrari, S. Khan, F. Castor Filho, and F. Dantas. Evolving software product lines with aspects: An empirical study on design stability. In *International Conference on Software Engineering (ICSE)*, pages 261–270, New York, NY, USA, 2008. ACM.

[46] S. Fischer, L. Linsbauer, R. E. Lopez-Herrejon, and A. Egyed. The ECCO tool: Extraction and composition for clone-and-own. In *37th International Conference on Software Engineering - Volume 2*, ICSE'15, pages 665–668, Piscataway, NJ, USA, 2015. IEEE Press.

[47] D. E. Goldberg, K. Deb, and J. H. Clark. Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, 6:333–362, 1992.

[48] M. Harman, Y. Jia, J. Krinke, B. Langdon, J. Petke, and Y. Zhang. Keynote: Search based software engineering for software product line engineering: a survey and directions for future work. In *18th International Software Product Line Conference*, pages 1–14, 2014.

[49] M. Harman and B. Jones. Search based software engineering. *Journal of Information and Software Technology*, 43(14):833–839, 2001.

[50] M. Harman, W. B. Langdon, and W. Weimer. Genetic programming for reverse engineering. In R. Oliveto and R. Robbes, editors, *20th Working Conference on Reverse Engineering (WCRE'13)*, Koblenz, Germany, 2013. IEEE. Invited Keynote.

[51] M. Harman, S. A. Mansouri, and Y. Zhang. Search based software engineering: A comprehensive analysis and review of trends techniques and applications. Technical Report Technical Report TR-09-03, Department of Computer Science, King's College London, 2009.

[52] M. Harman, S. A. Mansouri, and Y. Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR)*, 45(1):11:1–11:61, Dec. 2012.

[53] E. Haslinger, R. Lopez-Herrejon, and A. Egyed. Reverse engineering feature models from programs' feature sets. In *18th Working Conference on Reverse Engineering (WCRE)*, pages 308–312, Oct 2011.

[54] R. Heradio, H. Perez-Morago, D. Fernandez-Amoros, F. J. Cabrerizo, and E. Herrera-Viedma. A bibliometric analysis of 20 years of research on software product lines. *Information and Software Technology*, 72:1–15, 2016.

[55] C. Hofmeister, R. L. Nord, and D. Soni. Describing software architecture with UML. In *Working IFIP Conference on Software Architecture (WICSA)*, pages 145–159, Boston, MA, 1999. Springer US.

[56] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.

[57] M. Kelly, J. Alexander, B. Adams, and A. Hassan. Recovering a balanced overview of topics in a software domain. In *11th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 135–144, Sept 2011.

[58] J. Knodel, I. John, D. Ganesan, M. Pinzger, F. Usero, J. Arciniegas, and C. Riva. Asset recovery and their incorporation into product lines. In *12th Working Conference on Reverse Engineering (WCRE)*, pages 1–10, Nov 2005.

[59] C. W. Krueger. Software reuse. *ACM Computing Surveys (CSUR)*, 24(2):131–183, June 1992.

[60] C. W. Krueger. Easing the transition to software mass customization. In *Software Product-Family Engineering*, pages 282–293. Springer, 2002.

[61] U. Kulesza, V. Alves, A. Garcia, A. Neto, E. Cirilo, C. Lucena, and P. Borba. Mapping features to aspects: A model-based generative approach. In A. Moreira and J. Grundy, editors, *Early Aspects: Current Challenges and Future Directions*, volume 4765 of *Lecture Notes in Computer Science*, pages 155–174. Springer Berlin Heidelberg, 2007.

[62] K. Kumaki, R. Tsuchiya, H. Washizaki, and Y. Fukazawa. Supporting commonality and variability analysis of requirements and structural models. In *16th International Software Product Line Conference - Volume 2*, SPLC, pages 115–118, New York, NY, USA, 2012. ACM.

[63] M. A. Laguna and Y. Crespo. A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring. *Science of Computer Programming*, 78(8):1010–1034, 2013. Special section on software evolution, adaptability, and maintenance & Special section on the Brazilian Symposium on Programming Languages.

[64] S. Li, F. Chen, Z. Liang, and H. Yang. Using feature-oriented analysis to recover legacy software design for software evolution. In *International Conference on Software Engineering and Knowledge Engineering (SEKE 2005)*, pages 336–341, 2005.

[65] Y. Li, J. Yin, D. Shi, Y. Li, and J. Dong. Software product line oriented feature map. In Y. Shi, G. Albada, J. Dongarra, and P. Sloot, editors, *International Conference on*

Computational Science (ICCS), volume 4488 of *Lecture Notes in Computer Science*, pages 1115–1122. Springer Berlin Heidelberg, 2007.

[66] F. J. v. d. Linden, K. Schmid, and E. Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[67] L. Linsbauer, F. Angerer, P. Grünbacher, D. Lettner, H. Prähofer, R. E. Lopez-Herrejon, and A. Egyed. Recovering feature-to-code mappings in mixed-variability software systems. In *IEEE International Conference on Software Maintenance and Evolution*, pages 426–430, 2014.

[68] L. Linsbauer, E. R. Lopez-Herrejon, and A. Egyed. Recovering traceability between features and code in product variants. In *17th International Software Product Line Conference*, SPLC 2013, pages 131–140, New York, NY, USA, 2013. ACM.

[69] L. Linsbauer, R. E. Lopez-Herrejon, and A. Egyed. Feature model synthesis with genetic-programming. In *6th Symposium on Search Based Software Engineering*, 2014.

[70] R. E. Lopez-Herrejon, L. Linsbauer, W. K. Assunção, S. Fischer, S. R. Vergilio, and A. Egyed. Genetic improvement for software product lines: An overview and a roadmap. In *Conference on Genetic and Evolutionary Computation*, GECCO, pages 823–830, New York, NY, USA, 2015. ACM.

[71] F. Losavio, O. Ordaz, N. Levy, and A. Baiotto. Graph modelling of a refactoring process for product line architecture design. In *39th Latin American Computing Conference (CLEI'13)*, pages 1–12, 2013.

[72] A. Lozano. An overview of techniques for detecting software variability concepts in source code. In O. Troyer, C. Bauzer Medeiros, R. Billen, P. Hallot, A. Simitsis, and H. Mingroot, editors, *Workshops - Advances in Conceptual Modeling: Recent Developments and New Directions*, volume 6999 of *Lecture Notes in Computer Science*, pages 141–150. Springer Berlin Heidelberg, 2011.

[73] J. Martinez, T. Ziadi, T. F. Bissyandé, J. Klein, and Y. l. Traon. Automating the extraction of model-based software product lines from model variants. In *30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 396–406, 2015.

[74] J. Martinez, T. Ziadi, J. Klein, and Y. Traon. *10th European Conference Modelling Foundations and Applications*, chapter Identifying and Visualising Commonality and Variability in Model Variants, pages 117–131. Springer, 2014.

[75] S. Martinez-Fernandez, P. S. M. D. Santos, C. P. Ayala, X. Franch, and G. H. Travassos. Aggregating empirical evidence about the benefits and drawbacks of software reference architectures. In *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–10, 2015.

[76] M. Mefteh, N. Bouassida, and H. Ben-Abdallah. Feature model extraction from documented UML use case diagrams. *Ada User Journal*, 35(2):107–116, June 2014.

[77] A. Metzger and K. Pohl. Software product line engineering and variability management: Achievements and challenges. In *Future of Software Engineering*, FOSE 2014, pages 70–84, New York, NY, USA, 2014. ACM.

[78] E. Y. Nakagawa, P. O. Antonino, and M. Becker. Reference architecture and product line architecture: A subtle but critical difference. In *5th European Conference on Software Architecture*, ECSA'11, pages 207–211, Berlin, Heidelberg, 2011. Springer-Verlag.

[79] K. Nie, L. Zhang, and Z. Geng. Product line variability modeling based on model difference and merge. In *IEEE 36th Annual Computer Software and Applications Conference Workshops (COMPSACW)*, pages 509–513, 2012.

[80] A. Olszak and B. Nørregaard Jørgensen. Remodularizing Java programs for improved locality of feature implementations in source code. *Science of Computer Programming*, 77(3):131–151, 2012.

[81] V. Pareto. *Manuel D'Economie Politique*. Ams Press, Paris, 1927.

[82] L. Passos, K. Czarnecki, S. Apel, A. Wąsowski, C. Kästner, and J. Guo. Feature-oriented software evolution. In *7th International Workshop on Variability Modelling of Software-intensive Systems*, VaMoS 2013, pages 1–8, New York, NY, USA, 2013. ACM.

[83] X. Peng, Z. Xing, X. Tan, Y. Yu, and W. Zhao. Improving feature location using structural similarity and iterative graph mapping. *Journal of Systems and Software*, 86(3):664–676, 2013.

[84] B. Pfahringer. *Conjunctive Normal Form*, pages 209–210. Springer US, Boston, MA, 2010.

[85] K. Pohl, G. Böckle, and F. J. v. d. Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[86] A. Ramírez, J. R. Romero, and S. Ventura. A comparative study of many-objective evolutionary algorithms for the discovery of software architectures. *Empirical Software Engineering*, 21(6):2546–2600, 2016.

[87] C. Riva and C. Del Rosso. Experiences with software product family evolution. In *Sixth International Workshop on Principles of Software Evolution (IWPSE)*, pages 161–169, 2003.

[88] D. Romero, S. Urli, C. Quinton, M. Blay-Fornarino, P. Collet, L. Duchien, and S. Mosser. SPLEMMA: A generic framework for controlled-evolution of software product lines. In *5th International Workshop on Model-driven Approaches in Software Product Line Engineering; 4th Workshop on Scalable Modeling Techniques for Software Product Lines*, MAPLE/SCALE' 13, pages 59–66, New York, NY, USA, 2013.

[89] J. Rubin and M. Chechik. From products to product lines using model matching and refactoring. In *2nd International Workshop on Model-driven Approaches in Software Product Line Engineering (MAPLE '10), collocated with the 14th International Software Product Line Conference (SPLC '10)*, 2010.

[90] J. Rubin and M. Chechik. A survey of feature location techniques. In I. Reinhartz-Berger, A. Sturm, T. Clark, S. Cohen, and J. Bettin, editors, *Domain Engineering*, pages 29–58. Springer Berlin Heidelberg, 2013.

[91] J. Rubin, K. Czarnecki, and M. Chechik. Managing cloned variants: A framework and experience. In *17th International Software Product Line Conference*, SPLC 2013, pages 101–110, New York, NY, USA, 2013. ACM.

[92] P. Sampath. An elementary theory of product-line variations. *Formal Aspects of Computing*, pages 1–33, 2013.

[93] A. Santos, F. Gaia, E. Figueiredo, P. Santos Neto, and J. Araújo. Test-based SPL extraction: An exploratory study. In *28th Symposium on Applied Computing*, SAC 2013, pages 1031–1036, New York, NY, USA, 2013. ACM.

[94] S. Segura, J. Galindo, D. Benavides, J. A. Parejo, and A. R. Cortés. BeTTy: benchmarking and testing on the automated analysis of feature models. In *International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*, pages 63–71, 2012.

[95] S. She. *Feature Model Synthesis*. PhD thesis, University of Waterloo, Electrical and Computer Engineering Department, 2013.

[96] S. She, R. Lotufo, T. Berger, A. Wąsowski, and K. Czarnecki. Reverse engineering feature models. In *33rd International Conference on Software Engineering*, ICSE 2011, pages 461–470, New York, NY, USA, 2011. ACM.

[97] S. She, U. Ryssel, N. Andersen, A. Wąsowski, and K. Czarnecki. Efficient synthesis of feature models. *Information and Software Technology*, 56(9):1122–1143, 2014.

[98] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.

[99] C. Stoermer and L. O'Brien. MAP - Mining Architectures for Product Line Evaluations. In *Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 35–44, 2001.

[100] M. Svahnberg, J. van Gurp, and J. Bosch. A taxonomy of variability realization techniques: Research articles. *Software - Practice and Experience*, 35(8):705–754, July 2005.

[101] R. N. Taylor, N. Medvidovic, and E. M. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. Wiley Publishing, 2009.

[102] C. Thörn. *On the Quality of Feature Models*. PhD thesis, Linköping University, Department of Computer and Information Science, The Institute of Technology, 2010.

[103] M. Trifu. *Tool-supported identification of functional concerns in object-oriented code*. PhD thesis, Karlsruhe Institute of Technology, 2010.

[104] M. Völter, T. Stahl, J. Bettin, A. Haase, and S. Helsen. *Model-driven software development: technology, engineering, management*. John Wiley & Sons, 2013.

[105] S. Wang, S. Ali, T. Yue, Y. Li, and M. Liaaen. A practical guide to select quality indicators for assessing Pareto-based search algorithms in search-based software engineering. In *38th International Conference on Software Engineering*, ICSE '16, pages 631–642, New York, NY, USA, 2016. ACM.

[106] D. R. White. Software review: the ecj toolkit. *Genetic Programming and Evolvable Machines*, 13(1):65–67, 2012.

[107] Y. Xue. *Reengineering legacy software products into software product line*. PhD thesis, National University of Singapore, Department of Computer Science, 2012.

[108] Y. Xue, Z. Xing, and S. Jarzabek. Understanding feature evolution in a family of product variants. In *17th Working Conference on Reverse Engineering (WCRE)*, pages 109–118, Oct 2010.

[109] Y. Yang, X. Peng, and W. Zhao. Domain feature model recovery from multiple applications using data access semantics and formal concept analysis. In *16th Working Conference on Reverse Engineering (WCRE)*, pages 215–224, Oct 2009.

[110] Y. Yu, H. Wang, G. Yin, and B. Liu. Mining and recommending software features across multiple web repositories. In *5th Asia-Pacific Symposium on Internetware*, Internetware, pages 1–9, New York, NY, USA, 2013. ACM.

[111] G. Zhang, L. Shen, X. Peng, Z. Xing, and W. Zhao. Incremental and iterative reengineering towards software product line: An industrial case study. In *27th IEEE International Conference on Software Maintenance (ICSM)*, pages 418–427, 2011.

[112] T. Ziadi, C. Henard, M. Papadakis, M. Ziane, and Y. Le Traon. Towards a language-independent approach for reverse-engineering of software product lines. In *29th Symposium On Applied Computing (SAC)*, pages 1064–1071, March 2014.

Appendix A

Reengineering legacy applications into software product lines: a systematic mapping

Empir Software Eng
DOI 10.1007/s10664-017-9499-z

Reengineering legacy applications into software product lines: a systematic mapping

Wesley K. G. Assunção[1,2] · Roberto E. Lopez-Herrejon[3] · Lukas Linsbauer[4] · Silvia R. Vergilio[1] · Alexander Egyed[4]

Abstract Software Product Lines (SPLs) are families of systems that share common assets allowing a disciplined reuse. Rarely SPLs start from scratch, instead they usually start from a set of existing systems that undergo a reengineering process. Many approaches to conduct the reengineering process have been proposed and documented in research literature. This scenario is a clear testament to the interest in this research area. We conducted a systematic mapping study to provide an overview of the current research on reengineering of existing systems to SPLs, identify the community activity in regarding of venues and frequency of publications in this field, and point out trends and open issues that could serve as references for future research. This study identified 119 relevant publications. These

Communicated by: Per Runeson

✉ Wesley K. G. Assunção
wesleyk@inf.ufpr.br

Roberto E. Lopez-Herrejon
roberto.lopez@etsmtl.ca

Lukas Linsbauer
lukas.linsbauer@jku.at

Silvia R. Vergilio
silvia@inf.ufpr.br

Alexander Egyed
alexander.egyed@jku.at

[1] DInf, Federal University of Paraná (UFPR), CP: 19081, CEP: 81.531-980, Curitiba, Brazil

[2] COTSI, Federal University of Technology - Paraná (UTFPR), Cristo Rei Street, 19. CEP: 85.902-490, Toledo, Brazil

[3] Department of Software Engineering and IT, École de Technologie Supérieure, (ÉTS), Notre-Dame Street Ouest. 1100, H3C 1K3, Montreal, Canada

[4] ISSE, Johannes Kepler University Linz (JKU), Altenbergerstr. 69, 4040 Linz, Austria

Published online: 08 February 2017

primary sources were classified in six different dimensions related to reengineering phases, strategies applied, types of systems used in the evaluation, input artefacts, output artefacts, and tool support. The analysis of the results points out the existence of a consolidate community on this topic and a wide range of strategies to deal with different phases and tasks of the reengineering process, besides the availability of some tools. We identify some open issues and areas for future research such as the implementation of automation and tool support, the use of different sources of information, need for improvements in the feature management, the definition of ways to combine different strategies and methods, lack of sophisticated refactoring, need for new metrics and measures and more robust empirical evaluation. Reengineering of existing systems into SPLs is an active research topic with real benefits in practice. This mapping study motivates new research in this field as well as the adoption of systematic reuse in software companies.

Keywords Systematic reuse · Legacy systems · Evolution · Reengineering · Product family

1 Introduction

The premise of software reuse strategies is to use existing artefacts to build new software, aiming to reduce time-to-market, improving productivity and producing high quality software (Krueger 1992). According to Riva and Del Rosso, reuse is generally employed through an ad hoc strategy, called "clone-and-own methodology" (Riva and Del Rosso 2003). When customers request for additional features, existing systems are cloned and adapted to fulfill the new requirements. A main disadvantage of this methodology is the simultaneous maintenance of a typically large number of individual product variants (Faust and Verhoef 2003). In this scenario, as the number of features increases so does the complexity of their development and maintenance, hence a systematic reuse approach is necessary. A way to tackle this problem is the adoption of a *Software Product Line (SPL)* approach (Linden et al. 2007; Pohl and Böckle 2005).

An SPL is a set of systems that share common features, and are designed for a specific domain (Clements and Northrop 2001; Linden et al. 2007). The main advantage of an SPL is the systematic reuse of the common infrastructure – artefacts and assets – which is shared to create different product variants. *Software Product Line Engineering (SPLE)* is the discipline of developing and managing SPLs. SPLE identifies two types of assets: the *common assets*, reused in all products, and the *variable assets*, related to those features that are provided only by some products. In addition, SPLE deals with the effective management of SPLs throughout their entire life cycle. A way to undertake SPLE is by using existing product variants as the basis to create SPL assets, known as *extractive approach* (Krueger 2002).

Clone-and-own methodology was identified by a recent study as the most common reuse scenario in practice (Dubinsky et al. 2013). In this context, the extractive approach is the more common way to systematize the software reuse with SPLs and encompasses the reengineering of existing systems, leading to a systematic reuse and easier maintenance, because the systems are not maintained individually but instead as a group considering both common and variable assets. Besides these technical benefits, the reengineering of existing systems into an SPL allows companies to preserve their investment and aggregate knowledge obtained during the development of individual systems. Because of these

reasons, the extractive approach has attracted interest from companies, with many systems in production, and researchers (e.g. Lozano (2011)).

This increased interest prompted us to perform a *systematic mapping study*, an evidence-based method used to build a classification scheme and structure of a field of interest (Petersen et al. 2008, 2015). The goal of this study is threefold: (1) to provide an overview of the current research regarding the reengineering process of existing systems into SPLs, (2) identify the activity of the research community regarding the venues and frequency of publications in this field, and (3) point out research trends and gaps to direct future research on the subject.

This paper is an extension of our previous work (Assunção and Vergilio 2014). In our previous work we presented results of a systematic mapping on feature location and migration of existing systems to SPLs. We observed a consolidate research community in both areas and a strong relation between feature location and the reengineering process to migrate systems to SPLs. Despite extensive research, some phases on the reengineering process to undertake the migration task have not been fully investigated. Furthermore, we noticed the existence of a wide number of techniques adopted for the reengineering process and feature location. In this paper we make the following extensions to our previous work:

- *Inclusion of more primary sources*: we extended the range of dates, including two years of additional content, and the inclusion criteria, leading to 56 new pieces of work;
- *An enriched background*: more details about clone-and-own, software product lines, and the reengineering process are provided to support the readers that are new to the field;
- *Inclusion of new research questions and an extended classification schema*: eight research question are added and a classification schema with new dimensions is considered;
- *More in depth analysis*: the results are presented and analysed in more depth and detail. Such analysis takes into account the publication venues, the relation of the input/output artefacts with the strategies, and work on the intersection of different phases and strategies;
- *Detailed description of research avenues*: the main limitations of the existing approaches are pointed out. The goal is to synthesize evidence that suggests important implications for practice, and to identify challenges and areas for improvement.

The paper is organized as follows. Section 2 reviews the concepts of clone-and-own, software product lines, and the reengineering of systems to software product lines. Section 3 presents the methodology adopted in our mapping study. Section 4 describes the outcomes of the mapping process and discusses important findings and research opportunities that were identified. The threats to validity are presented in Section 5 and the related work in Section 6. Section 7 summarizes our conclusions.

2 Background

This section briefly reviews terminology and main concepts used throughout the paper.

2.1 Clone-and-Own

The most common scenario of reuse in practice is ad hoc techniques. For instance, when there exists demand for a new product that has some similar functionalities to an existing

product, usually developers fork the new product from other already existing software and then adapt it to fit the new requirements. These ad hoc practices are collectively called *Clone-and-Own (C&O)* (Dubinsky et al. 2013). Besides offering a simple and efficient way to reuse software artefact, products developed following C&O practices have their own and independent development life cycle. C&O approaches may work fine with small number of products, depending on products complexity, the development organization and its software engineering practices. However, in most situations, adding new systems is no longer doable either because of managerial, economical or technical reasons. For instance, the maintenance of many independent products leads to multiple problems like inefficient feature update or bug fixing, duplicate functionality, redundant and inadequate testing, etc (Dubinsky et al. 2013).

Variability is the capacity of software artefacts to vary. Besides the problems regarding maintenance of duplicated software artefacts, when we have to deal with a set of system variants another problem raises, known as *variability management*. The management of variability in a scenario with multiple system variants face several issues, i.e, extracting variability from technical artefacts, tool support, design decisions management and enforcement, testing of artefacts with variability, domain design, etc (Chen and Babar 2010; Metzger and Pohl 2014). The software product line approach is the premier alternative to cope effectively with the problems that emerge with the C&O practices, as discussed in next section.

2.2 Software Product Lines

Software Product Line (SPL) is "*a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission*" (Clements and Northrop 2001). Over the last two decades extensive research and practice have been done in the field of SPLs (Heradio et al. 2016). The benefits provided by SPL practices are better customization, improved software reuse, and faster time to market. The basis of the approach is that the products are built using a core asset base instead of being developed one by one from scratch (Heradio et al. 2016). Members of an SPL are distinguished by the set of features they provide (Pohl and Böckle 2005). A *feature* is "*a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems*" (Kang et al. 1990). Features are the building blocks of the products of an SPL.

Software Product Line Engineering (SPLE) is the discipline responsible to develop SPLs. SPLE exploits the *commonality* (i.e., a property shared by all products of an SPL) and the *variability* (i.e., the capacity of different applications of the product line to vary). An effective management and realization of variability is at the core of successful SPL development (Svahnberg et al. 2005). Krueger reported three ways used by companies as start point to SPLE (Krueger 2002):

- The *proactive approach*: first engineers perform a complete domain analysis, to have a full scope of products, then they develop reusable domain artefacts, and finally they use these artefacts to application engineering;
- The *reactive approach*: engineers incrementally grow their family of products applying both domain and application engineering every time a new product is developed;
- The *extractive approach*: engineers use existing custom software systems by extracting the common and varying artefacts, migrating them to an SPL.

The extractive approach is the most common way to adopt SPLs in companies with many software system variants in production (Krueger 2002). Besides, providing the benefits of

systematic reuse promoted by SPLs, the investment and knowledge necessary to develop the existing system are held.

2.3 Reengineering of Systems

According to Chikofsky and Cross, *reengineering* is "*the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form*" (Chikofsky and Cross J.HI 1990). Sometimes this term is confused with the term reverse engineering. However, the same authors define *reverse engineering* as "*the process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction.*" We can see that reverse engineering is concerned with understanding the systems, on the other hand, reengineering is concerned with restructuring/refactoring the systems. In this sense, reverse engineering is a prerequisite to the reengineering process, since we need to understand the subject system that we have to transform (Demeyer et al. 2009).

In the context of our work, the reengineering has focus on transforming a set of existing systems into an SPL. This set of existing systems can be reached by the use of ad hoc strategies of reuse, as mentioned in Section 2.1, or from a set of legacy systems. Reengineering of software systems into SPLs was the focus of two papers found in literature (Laguna and Crespo 2013; Fenske et al. 2013). These papers present a coarse-grained overview of the reengineering activity, answering questions about existing approaches, techniques, open challenges, and suggesting a taxonomy for existing approaches. From this point of view, our mapping study also has as goal the identification of approaches and techniques used for the reengineering process, but with focus on fine-grained details.

SPLE proposes activities to manage features, create variability models, and use variability mechanisms. However, taking into account the reengineering process, there are questions regarding the flow of reengineering phases, artefacts commonly used, tool support, etc., that remain unanswered. Recalling the goal of this paper, in this mapping we aim at exploring the literature to understand the reengineering process, and answer a set of research questions that remain open.

3 Systematic Mapping Process

Systematic mappings are studies designed to provide an overview of a research field and find research opportunities. After the search and selection of the relevant literature, the studies are classified and counted regarding categories of interest in the field (Petersen et al. 2008, 2015).

The study was carried out according to the mapping process proposed by Petersen et al. (2008, 2015), which includes the main activities: definition of research questions, conduction of the search and screening of papers, classification scheme, and data extraction and mapping. Each activity is described next.

3.1 Research Questions

As mentioned in the introduction, the goal of our systematic mapping is threefold: (1) to provide an overview of the current research on the field, (2) identify the publication venues and frequency, and (3) point out research trend and gaps for future work. Furthermore,

motivated by questions unanswered in related work, such as the flow of the reengineering process phases, input/output artefacts, and case studies used in evaluations; we formulated eight research questions grouped in three categories, as follows:

1. **Current research on reengineering systems into SPLs:**
   - *RQ1*: *What are the common phases of the reengineering process?* The goal of this question is to identify the common phases applied in the reengineering process. Furthermore, we can analyse the number of works devoted to each phase, and then to identify possible needs not addressed in the field;
   - *RQ2*: *What are the strategies used in the reengineering process?* In our context, a strategy is the application of a technique or method to obtain an SPL from existing systems. This question helps us to catalogue the strategies, techniques and methods, currently employed in the reengineering process;
   - *RQ3*: *What are the artefacts used as input and output?* A wide range of artefacts are produced along the software development process. In this question, we analyse what are the artefacts used for each strategy, considering both inputs and outputs;
   - *RQ4*: *What is the provenance of the systems used for the evaluation of the proposed approaches?* Our goal for this question is to obtain information regarding the provenance of the case studies used in the evaluations, for instance, academic or industrial systems. Based on this information, we can gauge at the maturity of the approaches in a specific scenario;
   - *RQ5*: *What are the tools available that support the reengineering process?* This question aims at cataloging the specific tools proposed and used to support the reengineering process. A list of existing tools can be used as reference for practitioners who need support for the reengineering process;
2. **Publications venues and frequency:**
   - *RQ6*: *Where has work been published?* There are few conferences and workshops devoted specifically to SPLs, but research on this subject can be published in different venues of Computer Science and Software Engineering. We want to know the most used fora to identify where the specialized community on this research topic has been publishing;
   - *RQ7*: *How have publication frequencies changed?* This question aims at analysing the evolution of the number of published papers on this research topic over the years. This information can help us to assess how relevant and active this topic is in the Software Engineering community.
3. **Trends and research opportunities:**
   - *RQ8*: *What are the research gaps and trends in the field of reengineering of systems variants into SPL?* This question aims at analyzing the limitation of existing approaches and identifying research directions for future work to motivate new research on this topic.

3.2 Conducting Search and Screening of Papers

In order to answer our research questions, the first step was the selection of relevant studies from the literature. To conduct the search of studies we must define a set of search terms. To reach a good set of terms we used a test-set of known papers that ought to be found. Using

Table 1 Search Terms

Feature Location Terms
”feature location”, “concept location”, “concern location”, “feature mining”, “feature identification”, “feature mapping”
Reengineering Terms
reengineering, refactoring, reconstruction, migration, migrating, evolution, legacy, restructuring, “re-engineering”, “re-structuring”
SPL Terms
”application engineering”, commonality, “core asset”, “domain analysis”, “domain engineering”, “feature analysis”, “feature based”, “feature diagram”, “feature model”, “feature modeling”, “feature oriented”, “highly-configurable system”, “process family”, “product family”, “product line”, “product line engineering”, “software family”, “software product family”, “software product line”, “software reuse”, SPL, variability, “variability analysis”, “variability management”,”variability modeling”, “variability-intensive system”, variant, variation, “variation point”

this reference set, we tried different combinations of keywords. In this way, we reached three groups.

The first group for terms regarding the location of features, which are the building blocks of SPLs. Feature location is the initial and crucial task to identify the functional parts of existing systems to be re-engineered into SPLS. The second group relates to the notion of reengineering of systems. For the second group we use terms similar to those presented in a related work (Laguna and Crespo 2013). The third group relates to common SPL terminology. For the last group, we employed SPL terms collected from twelve mapping studies on several aspects of SPLs that we used in a recent survey on Search-Based Software Engineering for SPLs (Lopez-Herrejon et al. 2015). The set of terms used to perform the search is presented in Table 1.[1]

The search query composed by these terms is as follows:[2]

(“feature location” OR “concept location” OR “concern location” OR “feature mining” OR “feature identification” OR “feature mapping”) AND (reengineering OR refactoring OR reconstruction OR migration OR migrating OR evolution OR legacy OR restructuring OR “re-engineering” OR “re-structuring”) AND (”application engineering” OR ”commonality” OR “core asset” OR “domain analysis” OR “domain engineering” OR “feature analysis” OR “feature based” OR “feature diagram” OR “feature model” OR “feature modeling” OR “feature oriented” OR “highly-configurable system” OR “process family” OR “product family” OR “product line” OR “product line engineering” OR “software family” OR “software product family” OR “software product line” OR “software reuse” OR “SPL” OR “variability” OR ”variability analysis” OR “variability management” OR”variability modeling” OR “variability-intensive system” OR ”variant” OR “variation” OR “variation point”)

[1] Alternative term spellings or upper/lower case are not shown in the table and were found not to be relevant for our searches.

[2] Some databases have a limit of character for the search string. In these cases the search query used was: `("feature location" OR "concept location" OR "concern location" OR "feature mining") AND (reengineering OR refactoring OR reconstruction OR migration OR migrating ) AND ("product line" OR "product-line" OR "product family" OR "program family")`

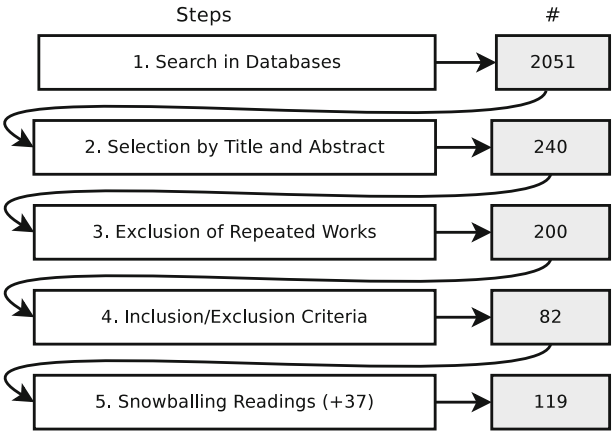


Fig. 1 Steps of the search and selection of the relevant papers

The search and selection of the relevant primary sources were conducted in five steps, shown in Fig. 1. In the first step we performed the search for relevant publications by using the search string presented above. The search started in 2014 when we prepared our previous work (Assunção and Vergilio 2014), then we performed the search again starting on January 4th 2016 and ended on February 12th 2016. We used seven academic databases, shown in Table 2. In the table the last column presents the number of studies found in each database. At the end of this step, we obtained a set of 2051 papers. In the second step, the title of the works was read to select the relevant ones. When the title was not enough to clarify the relevance of the paper, then the abstract was read. After reviewing the title and abstract of the 2051 papers, we selected 240 studies.

In the third step 40 duplicate publications found in different databases were discarded (see Fig. 1), remaining 200. In Step 4, we read the full length papers and the inclusion/exclusion criteria presented in Table 3 were applied. We designed a set of inclusion and exclusion criteria to select only studies that fit the goals of our study. In summary we selected only papers peer reviewed, in English, available online, and with focus on reengineering multiple systems to SPLs. Considering these criteria, a final set with 82 papers was obtained.

In the last step (Step 5) we performed snowballing readings. In the snowballing reading, citations and the reference list of found papers are used to identify other relevant studies

Table 2 Academic databases used in the mapping

Database	URL	#
Science Direct	http://www.sciencedirect.com	1080
Scopus	http://www.scopus.com	379
Web of Science	http://www.isiknowledge.com	7
IEEE Xplore	http://ieeexplore.ieee.org	3
ACM Digital Library	http://dl.acm.org	10
Springer	http://www.springerlink.com	333
Google Scholar	http://scholar.google.com	239

Table 3 Inclusion and exclusion criteria

Inclusion criteria
– Text in English;
– Publications in journals, conferences, workshops, abstracts, tutorials, short papers, tool demonstration, entire thesis, book chapter, and technical reports;
– Available in an electronic format: eps, DOC, HTML, etc;
– With focus on reengineering of SPLs starting from multiple systems.
Exclusion criteria
– Position papers, doctoral symposium;
– Mapping studies, surveys, state-of-art and literature review;
– Papers not available online;
– Without focus on reengineering and SPL;
– Starting the reengineering process with a single system.

(Wohlin 2014). We performed backward snowballing, that encompassed the use of the reference list of the 82 papers obtained in the previous step. We went through the reference lists looking for new relevant studies. For each paper identified for possible inclusion, we read the paper and applied the inclusion and exclusion criteria. Then, we identified 37 new relevant studies in this step. After this, the final set was composed of 119 papers.

The search and screening of papers discussed above were performed by the first and second authors. After composing the final set of primary sources the fourth author reviewed the included papers, hence assessed by an individual person.

3.3 Classification Scheme and Data Extraction

A classification scheme was used to guide the data extraction from relevant studies. Considering the goals and research questions of our mapping, there is no standard set of categories regarding the reengineering process known in literature. Considering this case we created our own classification scheme iteratively during the reading of the studies. The four steps to create the classification scheme were: (1) first we determined six dimensions related to the research questions about the reengineering process (RQ1 to RQ5), (2) then we read the primary sources, collected and documented all relevant concepts or terms taking into account the dimensions, (3) next the sets of concepts and terms from different papers were combined together, for that we analyzed which parts of the identified information were similar or common in different studies, and finally (4) one category for each similar/common item was created in the correlated dimension. This process was performed initially by the first author for our previous work (Assunção and Vergilio 2014) and refined after the search for new papers by the first three authors.

The dimensions defined about the reengineering process are: addressed reengineering phase (RQ1), type of technique or method applied (RQ2), artefact used as input for the process (RQ3), artefact generated as output (RQ3), type of systems used in the evaluation (RQ4), and existence of tool support (RQ5). The dimensions and categories, and a brief description of them are presented in Table 4. Each paper can belong to more than one category. Further details about the categories of each dimension are provided in the results.

Besides the classification scheme, we also captured complementary data regarding the reengineering process: what technique/method/algorithm was applied in each strategy

Table 4 Classification Scheme

Dimensions	Categories	Description
Reengineering Phase	Detection	Relevant information is extracted from the input artefacts, e.g. source code, to understand the existing structure, data flow, relationships, existing features, etc.
	Analysis	The information discovered is used to infer, design and organize new partitions that cluster the functional features.
	Transformation	When transformations are performed on the considered artefacts (such as source code) aiming at enabling the systematic reuse.
Strategy	Expert-driven	Strategy based on the expertise of specialists: software engineers, software architects, developers, stakeholders, etc.
	Static analysis	Static analysis relies on following or analysing structural information of static artefacts, in other words, without their execution (Wichmann et al. 1995).
	Dynamic analysis	When tools are used to collect and analyse information about the artefact's execution, in general considering a low-level of abstraction, such as source code (Cornelissen et al. 2009).
	Information Retrieval	This strategy leverages the fact that identifiers and comments represent domain knowledge. Commonly this strategy considers the textual similarity (Manning et al. 2008).
	Search-based	This strategy applies algorithms from the optimization field, such as Genetic Algorithms (Harman et al. 2009).
Input artefacts	Domain information	An example of this category is high level description of systems in specific domain and domain analysis.
	Requirements	Documents containing feature descriptions, customer requests, test sets generated, implementation and operation aspects, etc.
	Design models	Design artefacts include models such as class diagrams, state machines, or entity-relationship database model.
	Source code	Corresponds to the system implementation in a programming language.
Output artefacts	Features discovered	Features identified or mined from artefacts where they are not well-modularized or spread in multiple implementation units.
	Features mapped	Traceability links between known features and artefacts related with them, for instance, from requirements to source code.
	Reports	Reports with information such as the variability among the systems, impact on the reengineering to SPLs, and potential reuse in legacy system variants.
	Source code refactored	Source code refactored is an output provided to allow a better organization of the features with the SPLE.
Type of Systems	Industrial/Open source	Industrial or Open source systems are real case studies, developed by open source communities or by private companies. These systems vary from small to large systems.
	Academic/Illustrative	Academic or Illustrative systems, a.k.a. toy systems. Are generally small systems presented in text books or used to illustrate how approaches work.
	None	When the approaches do not use any system for their evaluation.
Tool support	Use of tool	When the study points the use of a tool to support the reengineering process.
	None	When the approaches do not use any tool to support the reengineering process.

category (RQ2), name of the systems used in the evaluation (RQ4), and name of the tools (RQ5). Furthermore, to answer other research questions the following data was also collected: paper title and publication venue (RQ6), publication year (RQ7), and approaches limitations, gaps, and future work mentioned (RQ8). In the next sections, the results and analysis of this classification are presented.

To assess the quality of the classification we proceed as described next. A subset of six papers from the the primary sources were selected and classified individually by the first three authors. Then the authors had a meeting to discuss the classifications. After agreeing regarding the classification, the first author classified the remaining papers. The final set of classifications was reviewed by the second and fourth authors.

4 Results

In this section, we describe the results of the data extraction phase. In Section 4.8 we present our analysis, identified trends, and research opportunities.

4.1 RQ1 - Phases of the Reengineering Process

In the context of obtaining SPLs from existing systems, there is not an established or widely accepted set of phases to conduct the reengineering process. Because of this, we inferred the phases using the data from the primary sources we identified. In most cases the phases of proposed approaches are not clearly described, so we inferred them also by considering the type of input and output artefacts. In summary, we observed that the main tasks that the approaches do are: (1) to identify the features existing in a set of systems or map features to their implementation, (2) to analyse the available artefacts and information to propose a possible SPL representation, and (3) to perform the modification in the artefacts to obtain the SPL. These phases are respectively called *detection*, *analysis*, and *transformation*, recalling the terminology proposed by Anwikar et al. (2012).

Based on the above mentioned, we created an overview of the reverse engineering process with focus on SPLs, which is presented in Fig. 2. Depicted on the left part of the figure we have the systems developed following C&O practices. The solid line represents

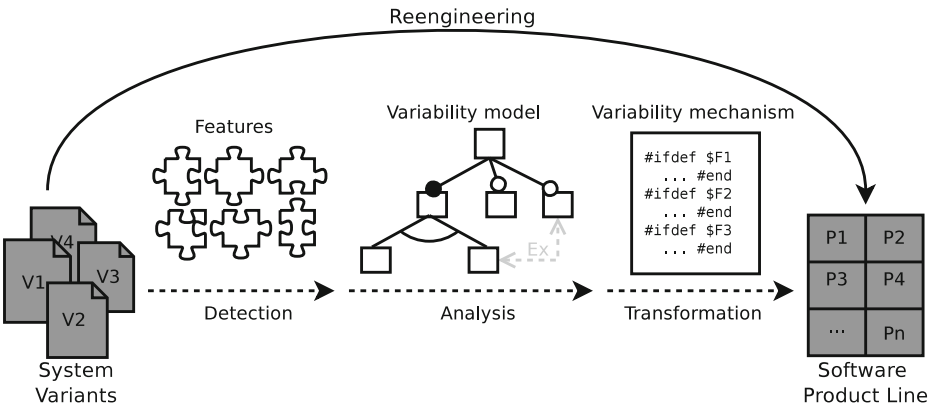


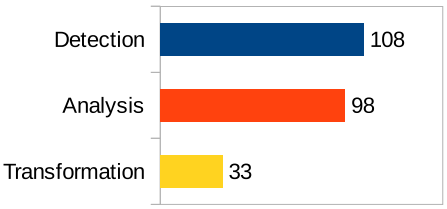
Fig. 2 Software reengineering process

the entire process of reengineering, however, it is usually composed by some phases, shown by dashed lines.

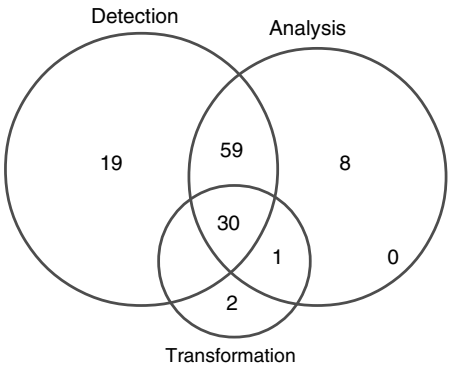
The generic phases of the reengineering process of existing systems into SPLs are: (1) *Detection phase* is the begining of the process, devoted to detecting the variability and commonality among existing products. Such as show in Fig. 2, the variabilities and commonalities are represented in terms of features. Common support in this phase is given by *feature location* techniques, which aim at locating the artifacts responsible for implementing the system functionalities. The management of the features and the mapping/traceability to the software artefacts that implement them are tasks of the SPLE domain engineering process (Dit et al. 2013; Rubin and Chechik 2013). (2) *Analysis phase* involves the organization of discovered variability and commonality. This step is devoted to creating the variability model, shown in the middle of Fig. 2, to express the valid combinations of features of an SPL. The feature model is the most popular form of variability model. Feature models are tree-like structures used to establish the existing relations between features (Kang et al. 1990). (3) *Transformation phase* is the last step of the process. Here artefacts that implement the features and the variability model are used to create the SPL, using a variability mechanism. For instance, the simplest mechanism is based on `#ifdef` statements made to the artefacts that are pre-processed, shown in the right of Fig. 2, following the desired feature selection, to create different products (Bachmann and Clements 2005). The reengineering can also be done considering design models (Wagner 2014).

Taking into account the phases of the reenginering process, Fig. 3a presents the distribution of studies among detection, analysis and transformation. We can observe that detection and analysis phases have received roughly the same attention. A possible reason for that is the existing relationship between them. When the detection is performed the analysis is a natural continuation. To extract information from artefacts without discovering helpful information for their reuse seems to make no sense. Consider that performing only detection without analysis would make sense for certain maintenance tasks (e.g. bug fixing), however, maintenance issues are outside of the scope of our mapping study. The transformation phase, which allows the actual systematic reuse of the artefacts, has not been extensively investigated.

Many proposed approaches have focus on more than one phase. Figure 3b presents the number of papers addressing each phase and the papers in more than one phase. As



a Primary sources per phase



b Intersection among the phases

Fig. 3 Reengineering phases

mentioned before, most papers involve detection and analysis (59), followed by papers that cover the three phases (30). Those papers with focus in only one phase are mainly in detection (19) and few ones in analysis (8) and transformation (2). Not surprisingly, we did not find papers with focus on detection and transformation only, because it does not make sense to perform these phases without adequate analysis. Table 5 presents the reference of studies according to the phases and their intersections.

Figure 4 presents a bubble chart with the number of the papers in each phase by year. Again, we can observe a correlation between detection and analysis. In spite of the lower

Table 5 Publications per phase

Phase	#	References
Detection	19	(Lohar et al. 2013; Falessi et al. 2010; Maia et al. 2008; Li et al. 2005; Stuikys and Valincius 2011; Knodel et al. 2005; Al-msie'deen et al. 2013; Heidenreich et al. 2008; Ferrari et al. 2013; Eisenbarth et al. 2001; Rubin and Chechik 2012b; Ziadi et al. 2012; Rubin et al. 2012; Eyal-Salman et al. 2013c; Van Der Storm 2007; Shao et al. 2013; Linsbauer et al. 2014; Niu et al. 2014; Guzman and Maalej 2014)
Analysis	8	(Linsbauer et al. 2013; Ryssel et al. 2011; Segura et al. 2012; Stoermer and O'Brien 2001; Linsbauer et al. 2014; Lopez-Herrejon et al. 2015; Eriksson et al. 2005; Eyal-Salman et al. 2014)
Transformation	2	(Romero et al. 2013; Mohamed et al. 2014)
Detect. + Analy.	59	(Schulze et al. 2013; Passos et al. 2013; Seidl et al. 2012; She et al. 2011; Koziolek et al. 2013; Anwikar et al. 2012; Eyal-Salman et al. 2012; Davril et al. 2013; Merschen et al. 2011; Xue et al. 2012; Damaševičius et al. 2012; Xue et al. 2010; AL-Msie'deen et al. 2012; Kelly et al. 2011; Nunes et al. 2013; Yang et al. 2009; Eyal Salman et al. 2013; Ziadi et al. 2014; Valinčius et al. 2013; Ali et al. 2011; Eyal-Salman et al. 2013b; Olszak and Jørgensen 2012; She et al. 2014; Peng et al. 2013; Gamez and Fuentes 2013; Sampath 2013; Alves et al. 2007; Li et al. 2007; Frenzel et al. 2007; Polzer et al. 2012; AL-Msie'deen et al. 2013; Kulesza et al. 2007; Almeida et al. 2006; Noor et al. 2008; Bécan 2013; Trifu 2010; She 2013; Acher et al. 2013; Haslinger et al. 2011; Eyal-Salman et al. 2013a; Martinez et al. 2014; Nöbauer et al. 2014a; Klatt et al. 2014; Acher et al. 2011; Nöbauer et al. 2014b; Gharsellaoui et al. 2015; Maazoun et al. 2014a; Mefteh et al. 2014; Maâzoun et al. 2014b; Abbasi et al. 2014; Alves et al. 2008; Weston et al. 2009; Chen et al. 2005; Acher et al. 2012; Hariri et al. 2013; Mu et al. 2009; Yu et al. 2013; Bagheri et al. 2012; Boutkova and Houdek 2011)
Analy. + Transf.	1	(Kolb et al. 2006)
Detect. + Analy. + Transf.	30	(Santos et al. 2013; Nunes et al. 2012; Rubin et al. 2013; Araújo et al. 2013; Knodel et al. 2005; Ramos and Penteado 2008; Duszynski et al. 2011; Klatt et al. 2013; Lago and Vliet 2004; Gamez and Fuentes 2011; Xue 2012; de Oliveira et al. 2012; Otsuka et al. 2011; Bayer et al. 2004; Bécan et al. 2013; Rubin and Chechik 2012a; Kang et al. 2005; Rubin and Chechik 2010; Losavio et al. 2013; Zhang et al. 2011; Breivold et al. 2008; Nie et al. 2012; Martinez et al. 2015; Rubin 2014; Fischer et al. 2015; Tang and Leung 2015; Fischer et al. 2014; Rubin et al. 2015; Faust and Verhoef 2003; Kumaki et al. 2012)

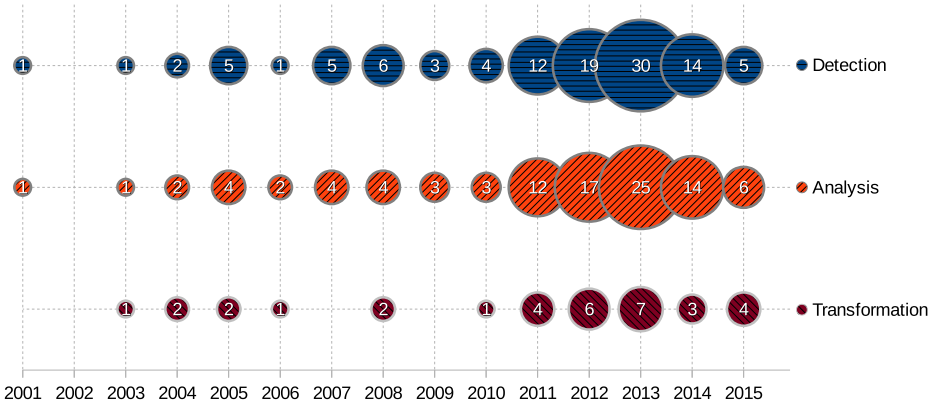


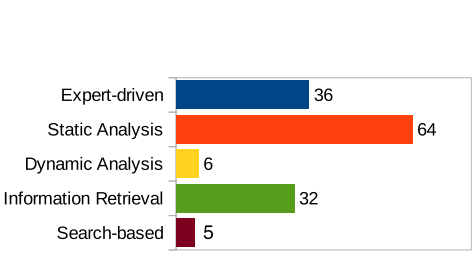
Fig. 4 Phases addressed by the studies per year

number of papers tackling transformation, we notice an interest in this phase over the years, mainly since 2011.

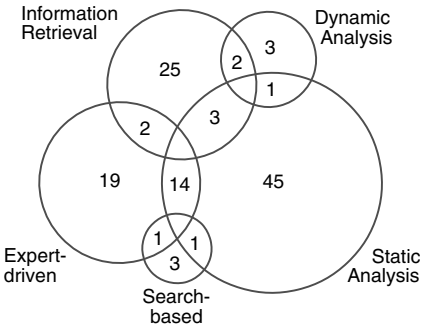
4.2 RQ2 - Strategies to Perform the Reengineering

According to our schema presented in Section 3, the strategies used in the reengineering were grouped in five categories. Figure 5a shows the number of papers in each strategy category. Static Analysis is the category with the largest number of papers. Expert-driven is the second most common strategy. Almost with the same attention appears Information Retrieval. Its preference is due to the ability to deal with documents/artefacts at a high level of abstraction, e.g. requirements in natural language.

We also observed that some studies do not use only one type of strategy, but instead use a combination of different strategies. This combination is sometimes named "hybrid" in literature (Rubin and Chechik 2013). For an overview about these combinations, Fig. 5b presents the number of studies in the intersections of different categories of strategies. Static analysis is the strategy most combined with other strategies, there are works using this strategy combined with all the other four strategies. Expert-driven and Information Retrieval are combined with other three strategies. Dynamic analysis and Search-based have studies



a Primary sources per strategy



b Intersection among the strategies

Fig. 5 Reengineering strategies

combined with only two other categories. The most frequent combination of strategies is Static analysis and Expert-driven, with 14 studies. Almost half of the primary sources (14 out of 36) that apply Expert-driven are combined with almost one fourth of studies (14 out of 64) that apply Static Analysis. Table 6 shows the references for the studies considering the intersections.

The classification of primary sources in each category and the techniques/methods found during the mapping are presented in Table 7. A technique or method is the concrete application of an algorithm, tool, or approach. In the fourth, fifth, and sixth columns of the table we present the percentage of works that deal with each phase. For example, in the category Expert-driven 32 out of 36 studies deal with the detection phase, 31 with analysis, and 17 with transformation.

The reengineering process conducted by experts (Expert-driven strategy) is the strategy that has the most number of works in the three phases of the reengineering process and has the largest number or works in transformation phase. Expertise seems to be adequate to perform the entire reengineering process. In the category of Static analysis we can observe that some techniques/methods also have a good number of studies in the three phases, for instance Heuristics and Overlaps; however, it does not happen for all techniques/methods of the strategy. Static analysis has the largest number of different techniques/methods, mainly dealing with detection and analysis. The strategy Dynamic analysis has only two methods that also have focus on the first two phases of the reengineering process. Information Retrieval has a good coverage of studies on the detection phase. This can be justified because of its ability to deal with large amount of data to discovery information. On the other hand the Search-based strategy deals mainly with analysis phase. We observed that search-based techniques have been used to create variability models that best represent existing systems, a complex activity.

Figure 6 shows the number of publications per year and type of strategy. The first strategies applied in 2001 were Expert-driven, Dynamic analysis and Information Retrieval. In 2004 the first work on Static Analysis appears, and recently in 2011, on a Search-based strategy. Until 2013 the number of research papers grew, addressing the strategies Expert-driven, Static Analysis, and Information Retrieval, but in the last two years the number of studies on these strategies decreased. The number of studies addressing the strategies Dynamic analysis and Search-based remains constant.

4.3 RQ3 - Input and Output Artefacts

In this section we summarize the results of the artefacts used. This summary is based on the type of artefact provided as input and produced as output by each paper, shown in Table 8. Regarding the input artefacts, Source code is the most common input artefact with 73 primary sources. Java, C, C++, and C# are the programming languages generally used. Requirements were the second most common with 55 primary sources. Examples of requirement artefacts are specifications, feature descriptions, customer requests, test suites, and documentation. Design models with 30 primary sources cover artefacts such as: class diagrams, state machines, and entity-relationship database models. Nine papers use Domain information, such as products description, user comments, documentation of systems in specific domain, and domain analysis.

Different types of artefacts are produced. We grouped them into four categories. Features mapped, the most common output with 34 studies, and Features discovered (27 primary sources) are in general outputs of the detection and analysis phases. When the features are known and well defined it is only necessary to obtain the mapping of features to the

Table 6 Publications per strategy

Strategy	#	References
Expert-driven	19	(Schulze et al. 2013; Santos et al. 2013; Passos et al. 2013; Koziolek et al. 2013; Rubin et al. 2013; Knodel et al. 2005; Ramos and Penteado 2008; Stuikys and Valincius 2011; Knodel et al. 2005; Almeida et al. 2006; Heidenreich et al. 2008; Lago and Vliet 2004; de Oliveira et al. 2012; Otsuka et al. 2011; Zhang et al. 2011; Stoermer and O'Brien 2001; Eriksson et al. 2005; Abbasi et al. 2014; Faust and Verhoef 2003)
Static Analysis	45	(Alves et al. 2007; Kulesza et al. 2007; Kelly et al. 2011; Valinčius et al. 2013; Chen et al. 2005; Yu et al. 2013; Hariri et al. 2013; Rubin et al. 2012; Nöbauer et al. 2014a; Klatt et al. 2014; Linsbauer et al. 2014; Xue et al. 2010; She et al. 2014; Trifu 2010; She 2013; Van Der Storm 2007; Losavio et al. 2013; Nie et al. 2012; Damaševičius et al. 2012; Nunes et al. 2012; She et al. 2011; Nunes et al. 2013; Bécan et al. 2013; Rubin and Chechik 2012a; Tang and Leung 2015; Bayer et al. 2004; Kang et al. 2005; Araújo et al. 2013; Romero et al. 2013; Seidl et al. 2012; Merschen et al. 2011; Gamez and Fuentes 2013; Polzer et al. 2012; Gamez and Fuentes 2011; Linsbauer et al. 2013; Duszynski et al. 2011; Martinez et al. 2014; Li et al. 2005; Acher et al. 2012; Frenzel et al. 2007; Mu et al. 2009; Haslinger et al. 2011; Rubin and Chechik 2012b; 2010; Peng et al. 2013)
Dynamic Analysis	3	(Anwikar et al. 2012; Maia et al. 2008; Olszak and Jørgensen 2012)
Information Retrieval	25	(Ryssel et al. 2011; Sampath 2013; AL-Msie'deen et al. 2013; Xue et al. 2012; AL-Msie'deen et al. 2012; Eyal-Salman et al. 2013b; Xue 2012; Gharsellaoui et al. 2015; Maazoun et al. 2014a; Maâzoun et al. 2014b; Eyal-Salman et al. 2013a, c; Mefteh et al. 2014; Ziadi et al. 2012; Niu et al. 2014; Eyal-Salman et al. 2012; Eyal Salman et al. 2013; Shao et al. 2013; Davril et al. 2013; Falessi et al. 2010; Ferrari et al. 2013; Guzman and Maalej 2014; Li et al. 2007; Kumaki et al. 2012; Ziadi et al. 2014)
Search-based	3	(Segura et al. 2012; Lopez-Herrejon et al. 2015; Linsbauer et al. 2014)
Exp. + Stat.	14	(Weston et al. 2009; Bécan 2013; Nöbauer et al. 2014b; Breivold et al. 2008; Martinez et al. 2015; Fischer et al. 2015; 2014; Mohamed et al. 2014; Noor et al. 2008; Acher et al. 2013; Kolb et al. 2006; Acher et al. 2011; Rubin 2014; Rubin et al. 2015)
Exp. + IR	2	(Boutkova and Houdek 2011; Bagheri et al. 2012)
Exp. + SB	1	(Lohar et al. 2013)
Stat. + Dyn	1	(Klatt et al. 2013)
Stat. + IR	3	(Eyal-Salman et al. 2014; Alves et al. 2008; Al-msie'deen et al. 2013)
Stat. + SB	1	(Ali et al. 2011)
Dyn. + IR	2	(Yang et al. 2009; Eisenbarth et al. 2001)

elements, commonly in source code. When these features are disorganized or spread across many code units, it is necessary to discover the features and its elements. 13 primary sources has as output Reports, which are often generated with information such as the variability among the systems, impact on the reengineering to SPLs, and potential reuse in legacy

Table 7 Strategies and methods used

Strategy	Technique/Method	#	# per Phase			References
			Det.	An.	Tr.	
Expert-driven	Expertise	36	32	31	17	(Schulze et al. 2013; Lohar et al. 2013; Santos et al. 2013; Passos et al. 2013; Koziolek et al. 2013; Rubin et al. 2013; Knodel et al. 2005; Ramos and Penteado 2008; Stuikys and Valincius 2011; Knodel et al. 2005; Almeida et al. 2006; Noor et al. 2008; Heidenreich et al. 2008; Bécan 2013; Lago and Vliet 2004; de Oliveira et al. 2012; Otsuka et al. 2011; Acher et al. 2013; Zhang et al. 2011; Stoermer and O'Brien 2001; Breivold et al. 2008; Kolb et al. 2006; Acher et al. 2011; Nöbauer et al. 2014b; Martinez et al. 2015; Rubin 2014; Eriksson et al. 2005; Fischer et al. 2015; Mohamed et al. 2014; Abbasi et al. 2014; Fischer et al. 2014; Rubin et al. 2015; Faust and Verhoef 2003; Weston et al. 2009; Bagheri et al. 2012; Boutkova and Houdek 2011)
Static Analysis	Clustering	13	12	12	0	(Weston et al. 2009; Kelly et al. 2011; Valinčius et al. 2013; Chen et al. 2005; Bécan 2013; Nöbauer et al. 2014b; Rubin et al. 2012; Damaševičius et al. 2012; Eyal-Salman et al. 2014; Alves et al. 2008; Hariri et al. 2013; Yu et al. 2013)
	Graph-based	12	12	11	3	(Chen et al. 2005; Klatt et al. 2014; Xue et al. 2010; Klatt et al. 2013; She et al. 2014; Trifu 2010; She 2013; Van Der Storm 2007; Losavio et al. 2013; Nie et al. 2012; Damaševičius et al. 2012; Peng et al. 2013)
	Heuristics	11	11	11	7	(Bécan 2013; Nöbauer et al. 2014b; Nunes et al. 2012; She et al. 2011; Nunes et al. 2013; Bécan et al. 2013; Rubin and Chechik 2012a; Tang and Leung 2015; Bayer et al. 2004; Kang et al. 2005; Araújo et al. 2013)
	Overlaps	11	10	9	6	(Martinez et al. 2015; Fischer et al. 2015; 2014; Linsbauer et al. 2013; Duszynski et al. 2011; Martinez et al. 2014; Linsbauer et al. 2014; Haslinger et al. 2011; Rubin 2014; Rubin et al. 2015; Rubin and Chechik 2012b)

Table 7 (continued)

Strategy	Technique/Method	#	# per Phase			References
			Det.	An.	Tr.	
	Structural Similarity	10	9	8	4	(Noor et al. 2008; Acher et al. 2013; Kolb et al. 2006; Acher et al. 2011; Al-msie'deen et al. 2013; Rubin and Chechik 2010; Peng et al. 2013; Rubin 2014; Rubin et al. 2015; Rubin et al. 2012)
	Model Transformation	8	7	7	3	(Romero et al. 2013; Kulesza et al. 2007; Araújo et al. 2013; Seidl et al. 2012; Merschen et al. 2011; Gamez and Fuentes 2013; Polzer et al. 2012; Gamez and Fuentes 2011)
	Dependency Analysis	5	5	4	1	(Breivold et al. 2008; Ali et al. 2011; Nöbauer et al. 2014a; Klatt et al. 2014; Linsbauer et al. 2014)
	Rule-based	3	2	2	1	(Hariri et al. 2013; Mohamed et al. 2014; Mu et al. 2009)
	Aspect Programming	2	2	2	0	(Alves et al. 2007; Kulesza et al. 2007)
	Data Flow Analysis	2	2	2	2	(Bayer et al. 2004; Kang et al. 2005)
	Program slicing	1	1	0	0	(Li et al. 2005)
	Propositional logic	1	1	1	0	(Acher et al. 2012)
	Reflexion Method	1	1	1	0	(Frenzel et al. 2007)
Dynamic Analysis	Execution Tracing	5	5	3	1	(Klatt et al. 2013; Anwikar et al. 2012; Maia et al. 2008; Olszak and Jørgensen 2012; Eisenbarth et al. 2001)
	Data Access Semantics	1	1	1	0	(Yang et al. 2009)
Information Retrieval	Formal Concept Analysis	17	15	14	1	(Eisenbarth et al. 2001; Yang et al. 2009; Eyal-Salman et al. 2014; Ryssel et al. 2011; Sampath 2013; AL-Msie'deen et al. 2013; Al-msie'deen et al. 2013; Xue et al. 2012; AL-Msie'deen et al. 2012; Eyal-Salman et al. 2013b; Xue 2012; Gharsellaoui et al. 2015; Maazoun et al. 2014a; Maâzoun et al. 2014b; ; Eyal-Salman et al. 2013a, c Mefteh et al. 2014)

Table 7 (continued)

Strategy	Technique/Method	#	# per Phase			References
			Det.	An.	Tr.	
	Latent Semantic Indexing	14	14	11	1	(Al-msie'deen et al. 2013; Xue et al. 2012; AL-Msie'deen et al. 2012; Eyal-Salman et al. 2013b; Xue 2012; Gharsellaoui et al. 2015; Maazoun et al. 2014a; Maâzoun et al. 2014b; Eyal-Salman et al. 2013a, c; Eyal-Salman et al. 2012; Eyal Salman et al. 2013; Shao et al. 2013; Alves et al. 2008)
	Other Natural Language Processing techniques	7	7	3	0	(Mefteh et al. 2014; Boutkova and Houdek 2011; Falessi et al. 2010; Ferrari et al. 2013; Guzman and Maalej 2014; Bagheri et al. 2012; Niu et al. 2014)
	Vector Space Model	4	4	3	1	(Eyal-Salman et al. 2013a, c; Kumaki et al. 2012; Alves et al. 2008)
	Word Frequency	2	2	1	0	(Ziadi et al. 2012; Ziadi et al. 2014)
	Data Mining	2	2	1	0	(Davril et al. 2013; Guzman and Maalej 2014)
	Ontology	2	2	2	0	(Bagheri et al. 2012; Li et al. 2007)
Search-based	Genetic Algorithm	3	1	2	0	(Lohar et al. 2013; Segura et al. 2012; Lopez-Herrejon et al. 2015)
	Genetic Programming	1	0	1	0	(Linsbauer et al. 2014)
	Non-dominated Genetic Algorithm II	1	1	1	0	(Ali et al. 2011)
	Hill climbing	1	0	1	0	(Lopez-Herrejon et al. 2015)
	Random search	1	0	1	0	(Lopez-Herrejon et al. 2015)

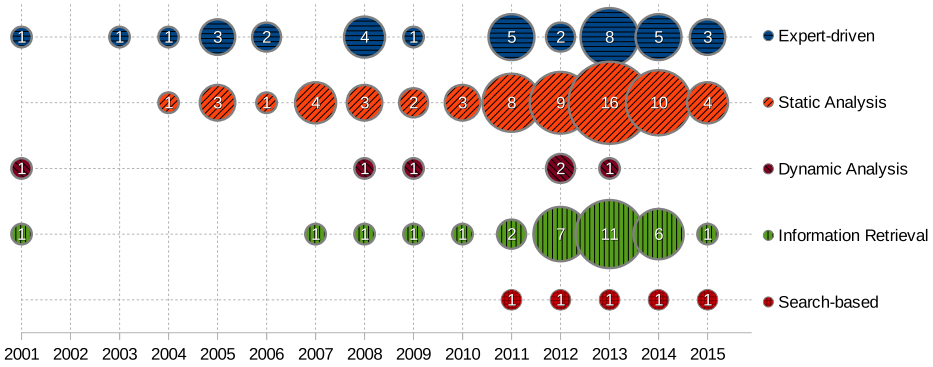


Fig. 6 Strategies addressed by the studies per year

system variants. Source code refactored, the second more generated output with 31 studies, is a common output of the transformation phase. After generating a feature-to-code traceability, the source-code elements associated to a feature can be: clustered into a Java package (in case of object-oriented programming, e.g. Olszak and Jørgensen (2012)), migrated to an aspect (in aspect-oriented programming, e.g. Alves et al. (2007)), or reformulated as components assets (e.g. Knodel et al. (2005)).

To analyse the relationship between strategies and inputs/outputs artefacts we use Fig. 7, which shows a bubble chart that maps the studies considering both dimensions. Expert-driven, Static analysis and Information Retrieval are strategies that have primary sources that use the four categories of input and output. In these strategies the most common input is the Source code (26, 38, and 20, respectively), and the most common output is the Source code refactored (15) for Expert-drive and Features discovered for Static analysis (30) and Information Retrieval (17). Dynamic analysis does not use Domain information as input and it also does not have primary sources that generate Reports. Finally, the Search-based strategy has primary sources that use as input Requirements, Design models and Source code, and generate only Features discovered and Reports.

Studies of all strategies have as input Requirements, Design models and Source code, but only studies based on Expert-driven, Static Analysis and Information Retrieval strategies use Domain information. For all strategies there are studies producing as output Features discovered and Source code refactored. About other outputs, only the Search-based strategy does not generate Features mapped. Moreover, Dynamic Analysis and Search-based strategies do not generate Reports, what is expected, since the studies with these strategies do not use Domain artifacts as input.

4.4 RQ4 - Systems Used for the Evaluation

We also analysed if the approaches proposed were evaluated and what kind of systems were used. Our results are summarized in Fig. 8. Most primary sources (57) use industrial/open source systems in the evaluations of their studies; 44 studies use academic/illustrative systems. 12 studies use both types of systems. In these cases, academic/illustrative systems are generally used in a controlled experiment and industrial/open source systems for a better evaluation. None type of evaluation was found in six of the papers. The systems used vary in the domain and size. Table 9 presents the main systems used in the evaluations.

4.5 RQ5 - Tool Support for the Reengineering Process

We present a summary of the tools proposed in the collected papers and used for evaluation. An total of 19 tools for the reengineering support were found. We only consider tools that are specific for the reengineering process. A brief description of the tools, and corresponding work references are presented in Table 10. The first tool appeared in 2007 (Alves et al. 2007)

Table 8 Categories of inputs and outputs artefacts used

Category	#	References
Input		
Domain information	9	(Bagheri et al. 2012; Almeida et al. 2006; Davril et al. 2013; Hariri et al. 2013; Acher et al. 2012; Yu et al. 2013; Ferrari et al. 2013; Mefteh et al. 2014; Guzman and Maalej 2014)
Requirements	55	(Almeida et al.2006; Koziolek et al. 2013; Knodel et al. 2005; Ramos and Penteado 2008; Noor et al. 2008; Lago and Vliet 2004; Kolb et al. 2006; Maia et al. 2008; Losavio et al. 2013; Eyal-Salman et al. 2014; Stuikys and Valincius 2011; Linsbauer et al. 2013; Eyal Salman et al. 2013; Eyal-Salman et al. 2013b; Xue 2012; Eisenbarth et al. 2001; Eyal-Salman et al. 2013c; Van Der Storm 2007; Eyal-Salman et al. 2013a; She 2013; She et al. 2011; Haslinger et al. 2011; Linsbauer et al. 2014; Lopez-Herrejon et al. 2015; Nunes et al. 2012, 2013; Ryssel et al. 2011; Ali et al. 2011; Bécan 2013; Bécan et al. 2013; Rubin et al. 2013; Araújo et al. 2013; Falessi et al. 2010; Valinčius et al. 2013; Li et al. 2007; Trifu 2010; Shao et al. 2013; Nöbauer et al. 2014a; Martinez et al. 2015; Rubin 2014; Faust and Verhoef 2003; Kumaki et al. 2012; Niu et al. 2014; Alves et al. 2008; Weston et al. 2009; Chen et al. 2005; Mu et al. 2009; Boutkova and Houdek 2011; Merschen et al. 2011; Polzer et al. 2012; Rubin et al. 2015; Stoermer and O'Brien 2001; Santos et al. 2013; Mefteh et al. 2014; Eriksson et al. 2005)
Design models	30	(Faust and Verhoef 2003; Acher et al. 2013; 2011; Xue 2012; Kumaki et al. 2012; Lago and Vliet 2004; Knodel et al. 2005; Yang et al. 2009; Romero et al. 2013; Kulesza et al. 2007; Passos et al. 2013; Seidl et al. 2012; Eyal-Salman et al. 2012; Peng et al. 2013; Mohamed et al. 2014; Gamez and Fuentes 2011; Heidenreich et al. 2008; Segura et al. 2012; Li et al. 2005; Rubin and Chechik 2010; Knodel et al. 2005; Li et al. 2007; Martinez et al. 2015; Rubin 2014; Rubin et al. 2015; Schulze et al. 2013; Xue et al. 2010; Martinez et al. 2014; Bécan 2013; Nie et al. 2012)
Source code	73	(Almeida et al. 2006; Faust and Verhoef 2003; Li et al. 2007; Eisenbarth et al. 2001; Nöbauer et al. 2014b; Lohar et al. 2013; Gharsellaoui et al. 2015; Stuikys and Valincius 2011; de Oliveira et al. 2012; Van Der Storm 2007; Damaševičius et al. 2012; Olszak and Jørgensen 2012; Kang et al. 2005; Sampath 2013; Tang and Leung 2015; Breivold et al. 2008; Alves et al. 2007; Seidl et al. 2012; Mohamed et al. 2014; Noor et al. 2008; Linsbauer et al. 2013; Xue et al. 2012; Klatt et al. 2013; Frenzel et al. 2007; AL-Msie'deen et al. 2013; Otsuka et al. 2011; Zhang et al. 2011; Maazoun et al. 2014a; Fischer et al. 2014; Eyal Salman et al. 2013; Ziadi et al. 2012; Bayer et al. 2004; She et al. 2014; Gamez and Fuentes 2013; Xue 2012; Kelly et al. 2011; Bécan 2013; Rubin and Chechik 2012a; Rubin et al. 2012; Lago and Vliet 2004; Passos et al. 2013; Eyal-Salman et al. 2012; Peng et al. 2013; Knodel et al. 2005; Koziolek et al. 2013; Ramos and Penteado 2008; Eyal-Salman et al. 2013a, b, c; Nunes et al. 2012; Rubin et al. 2013; Valinčius et al. 2013; Shao et al. 2013; Nöbauer et al. 2014a; Polzer et al. 2012; Santos et al. 2013; Al-msie'deen et al. 2013; Rubin and Chechik 2012b; Klatt et al. 2014; Trifu 2010; Anwikar et al. 2012; Martinez et al. 2015; Rubin 2014; Rubin et al. 2015; Eyal-Salman et al. 2014; Maâzoun et al. 2014b; Duszynski et al. 2011; AL-Msie'deen et al. 2012; Kolb et al. 2006; Ziadi et al. 2014; Fischer et al. 2015; Linsbauer et al. 2014; Abbasi et al. 2014)

Table 8 (continued)

Category	#	References
Output		
Features mapped	34	(Rubin et al. 2015; Fischer et al. 2014, 2015; Eyal-Salman et al. 2014; Martinez et al. 2015; Hariri et al. 2013; Rubin 2014; Boutkova and Houdek 2011; Eisenbarth et al. 2001; Romero et al. 2013; Schulze et al. 2013; Eyal-Salman et al. 2013a, b; Heidenreich et al. 2008; Lago and Vliet 2004; Van Der Storm 2007; Eyal-Salman et al. 2013c; Polzer et al. 2012; Anwikar et al. 2012; Linsbauer et al. 2014; Kulesza et al. 2007; Nunes et al. 2013; Seidl et al. 2012; Trifu 2010; Eriksson et al. 2005; Shao et al. 2013; Eyal-Salman et al. 2012; Merschen et al. 2011; Stuikys and Valincius 2011; Linsbauer et al. 2013; Passos et al. 2013; Peng et al. 2013; Ziadi et al. 2014; Eyal Salman et al. 2013)
Features discovered	27	(Frenzel et al. 2007; Xue et al. 2012; Maia et al. 2008; Almeida et al. 2006; Rubin and Chechik 2010; Ferrari et al. 2013; Falessi et al. 2010; Martinez et al. 2015; Eriksson et al. 2005; Damaševičius et al. 2012; Sampath 2013; AL-Msie'deen et al. 2013; Maazoun et al. 2014a; She et al. 2014; Bécan 2013; Valinčius et al. 2013; Maâzoun et al. 2014b; Abbasi et al. 2014; Acher et al. 2013; 2011; Kumaki et al. 2012; Yang et al. 2009; Li et al. 2005; Xue et al. 2010; She 2013; Linsbauer et al. 2014; Lopez-Herrejon et al. 2015)
Reports	13	(Merschen et al. 2011; Bagheri et al. 2012; Rubin 2014; Zhang et al. 2011; Guzman and Maalej 2014; Yu et al. 2013; Niu et al. 2014; Mu et al. 2009; Knodel et al. 2005; Koziolek et al. 2013; Noor et al. 2008; Stoermer and O'Brien 2001; Martinez et al. 2015)
Source code refactored	31	(Santos et al. 2013; Alves et al. 2007; Knodel et al. 2005; Tang and Leung 2015; Kolb et al. 2006; Ali et al. 2011; Olszak and Jørgensen 2012; Nunes et al. 2012; Martinez et al. 2015; Fischer et al. 2014; 2015; Rubin 2014; de Oliveira et al. 2012; Klatt et al. 2013; Rubin and Chechik 2012a; Rubin et al. 2015; Faust and Verhoef 2003; Mohamed et al. 2014; Kumaki et al. 2012; Breivold et al. 2008; Bayer et al. 2004; Losavio et al. 2013; Gamez and Fuentes 2011; Rubin et al. 2013; Kang et al. 2005; Otsuka et al. 2011; Xue 2012; Ramos and Penteado 2008; Maazoun et al. 2014a; Gharsellaoui et al. 2015; Kelly et al. 2011)

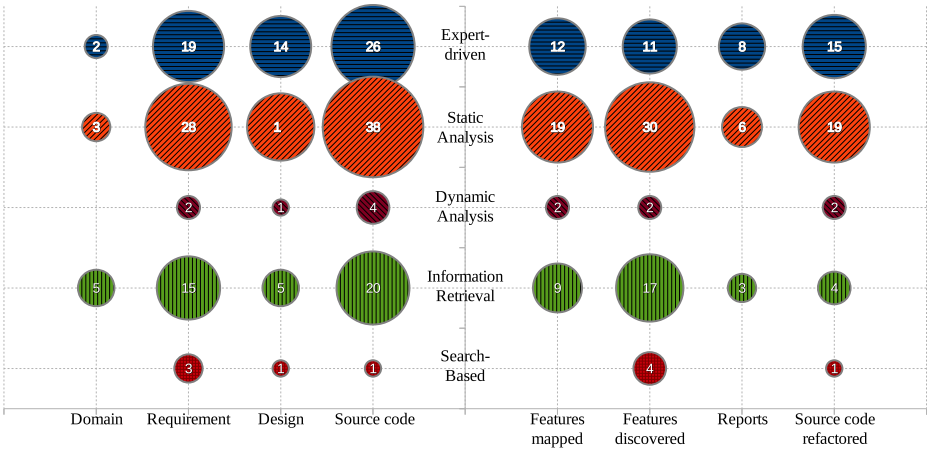


Fig. 7 Input and output used by the strategies

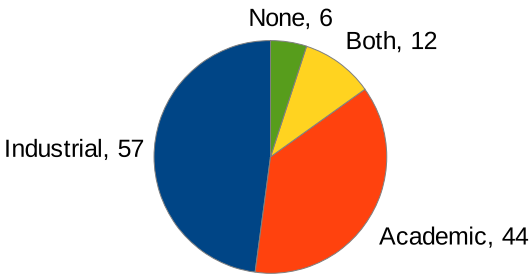


Fig. 8 Number of studies per type of system used in the evaluation

and since then, papers that present tools has become more common. This was expected, because of the increasing number of papers on the subject.

Table 11 presents the phases, strategies, input and output per tool. We can observe that most tools have focus on phases of detection and analysis, usually applying the Static analysis strategy. For Dynamic analysis and Search-based strategies, we found only one tool. Eight tools cover the three phases of the reengineering process. Regarding the input and output, the majority of tools use source code as input, followed by requirements and design artifacts. Considering that source code is the most common artifact, it is expected that refactoring of source code is the most common output. The mapping and/or discovering of features in system variants are also well covered by the tools. There are only two tools that generate reports to support SPL adoption.

4.6 RQ6 - Type and Fora of the Publications

First, we analyse the fora of the publications. The number of studies in each category is presented in Fig. 9. We can see that most papers were published in peer reviewed venues: journals, conferences, and workshops (∼92 % = 110 papers). This indicates that the area is being disseminated through a wide range of scientific outlets.

Conferences are the most frequent publication venue, followed by journals and workshops. The papers in these three types are distributed among 63 distinct venues: 17 journals, 38 conferences, and 8 workshops. Figure 10 shows the main fora with the most number of publications. From the 63 publication venues only 18 have two or more publications, and are accounting for a total of 65 papers, or 55 %. International Software Product Line Conference (SPLC) and Working Conference on Reverse Engineering (WCRE) are specific conferences and the preferred ones. Together they published 24 papers (20 %) of the primary sources. Information and Software Technology, Journal of Systems and Software, and Software: Practice and Experience are the journals with the largest number of publications. All of them are general Software Engineering publication venues. The main workshops are International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS), International Workshop on Model-driven Approaches in Software Product Line Engineering (MAPLE) and International Workshop on Reverse Variability Engineering (REVE).

The distribution of the number of publications does not correlate with the number of distinct venues. For example, 52 (∼67 %) conference publications appear in 12 (∼32 %) distinct venues, while all the 20 journal papers appear in 17 distinct venues. This is maybe due to the small number of conferences devoted to SPLs and maintenance/evolution, which are

Table 9 Systems used in evaluations

Industrial/Open source
ArgoUML (Eyal-Salman et al. 2013a; Linsbauer et al. 2014; Klatt et al. 2014; Ziadi et al. 2012; Al-msie'deen et al. 2013; Linsbauer et al. 2013; Eyal Salman et al. 2013; Eyal-Salman et al. 2013b; AL-Msie'deen et al. 2013); Eyal-Salman et al. 2013c, 2014; Fischer et al. 2014), Linux kernel (She et al. 2011; Xue et al. 2012; Peng et al. 2013; Xue 2012; She 2013), Softpedia Repository (Hariri et al. 2013; Yu et al. 2013; Guzman and Maalej 2014; Bagheri et al. 2012) Eclipse Project and Plugins (Knodel et al. 2005; Bayer et al. 2004; Martinez et al. 2015), Berkeley DB (Linsbauer et al. 2014; Xue 2012; Nie et al. 2012), Smart Home (Araújo et al. 2013; Alves et al. 2008; Weston et al. 2009), Automarker systems (Niu et al. 2014; Boutkova and Houdek 2011), JHotDraw (Trifu 2010; Olszak and Jørgensen 2012), Prevayler (Linsbauer et al. 2014; Tang and Leung 2015), FraSCAti (Acher et al. 2013; 2011), Microsoft Dynamics AX (Nöbauer et al. 2014b; 2014a), Defense domain systems (Rubin et al. 2015; Eriksson et al. 2005), Gantt Project (Noor et al. 2008), BlueJ (Olszak and Jørgensen 2012), Apache Web Server (Linsbauer et al. 2014), CCHIT Health (Lohar et al. 2013), CM-1 NASA (Lohar et al. 2013), Communications-Based Train Control (CBTC) (Ferrari et al. 2013), Curl (Linsbauer et al. 2014), DesktopSearcher (Linsbauer et al. 2014), E-Clinic (Lohar et al. 2013), eCos kernel (She et al. 2011), FreeBSD (She et al. 2011), Fujitsu Kyushu Network Technologies (Otsuka et al. 2011), Health watcher (Al-msie'deen et al. 2013), I-Trust (Lohar et al. 2013), Alcatel-Lucent IXM-PF (Zhang et al. 2011), Image Memory Handler (IMH) (Kolb et al. 2006), Java Buffer Library (Damaševičius et al. 2012), Jforum (Yang et al. 2009), JGossip (Yang et al. 2009), Labor Market Monitoring Software Product Line (LMMSPL) (Shao et al. 2013), LLVM Compiler (Linsbauer et al. 2014), MVNForum (Yang et al. 2009), Pooka Email Client (Ali et al. 2011), Power control and protection system (Breivold et al. 2008), Printworks (Li et al. 2005), QlikView (Nöbauer et al. 2014b), SELEX Sistemi Integrati Systems (Falessi et al. 2010), SIP Communicator (Ali et al. 2011), WebStore (Santos et al. 2013), Wget (Linsbauer et al. 2014), x264 Library (Linsbauer et al. 2014), Traffic management systems (Niu et al. 2014), Collaborative Software Suite (CoSS) (Hariri et al. 2013), Sudoku (Tang and Leung 2015), Web Product configurators (Abbasi et al. 2014), KePlast platform (Linsbauer et al. 2014), GameOfLife (Fischer et al. 2014), Electric motor controller (Rubin et al. 2015), Global trading and settlement system (GTSS) (Faust and Verhoef 2003)
Academic/Illustrative
MobileMedia (Al-msie'deen et al. 2013; Eyal-Salman et al. 2013b; Linsbauer et al. 2013; Eyal-Salman et al. 2012, 2014; Mefteh et al. 2014; Tang and Leung 2015), Mobile Phone (Araújo et al. 2013; Nie et al. 2012; Li et al. 2007; Maazoun et al. 2014a; Gharsellaoui et al. 2015), SPLOT Feature Models (She et al. 2014; Bécan 2013; Bécan et al. 2013; Haslinger et al. 2011; Acher et al. 2012; Lopez-Herrejon et al. 2015), Graph Product Line (GPL) (Linsbauer et al. 2014; Ziadi et al. 2012), Video On Demand (Linsbauer et al. 2013; Fischer et al. 2014), Wingsoft Financial Management System (Xue et al. 2010; Xue 2012), Banking System (Martinez et al. 2014; Maâzoun et al. 2014b), ZipMe (Linsbauer et al. 2014; Fischer et al. 2014), Washing Machine (Rubin and Chechik 2010; Rubin 2014), Airbag (Schulze et al. 2013), DirectBank (Peng et al. 2013), Home Automation (Nie et al. 2012), Home Service Robot (Kang et al. 2005), Insurance Polocy (Nie et al. 2012), J2ME Games Product Line (Kulesza et al. 2007), LinkedList (Linsbauer et al. 2014), PKJab (Linsbauer et al. 2014), Project Factory (Noor et al. 2008), SensorNetwork (Linsbauer et al. 2014), Graphical Editor (Maia et al. 2008), Text Editing System SPL (AL-Msie'deen et al. 2012), Vending Machine (Martinez et al. 2014), Microwave Oven (Rubin 2014), Xfig System (Eisenbarth et al. 2001), Fame-DBMS (Linsbauer et al. 2014), Jbook (Santos et al. 2013), Notepad SPL (Ziadi et al. 2014), Software Design Robot (Kumaki et al. 2012), Library management systems (Chen et al. 2005), Wiki engines (Acher et al. 2012), Suppliers offering systems (Acher et al. 2012), ModelAnalyzer (Fischer et al. 2014), Draw Product Line (Fischer et al. 2014), PCM Dataset (Bécan et al. 2013)

the preferred ones in this category (conferences). On the other hand, the journals with related publications are with general purpose on Software Engineering and Computer Science. In this category, there is a great number of possibilities (journals) that could be chosen.

4.7 RQ7 - Number and Frequency of Publications

The evolution on the number of publications along the years is depicted in Fig. 11. The first papers appeared in 2001 and there was an increase in the number of publications in 2005

Table 10 Tools used in the reengineering process

Name	Reference	Description
Variability to Aspect tool	(Alves et al. 2007)	Aims at extracting variations from existing products by isolating such variations into aspects.
FeatureMapper	(Heidenreich et al. 2008; Seidl et al. 2012)	A tool that allows defining mappings of features to model elements, specifying feature realisations.
CoDEx Tool	(Trifu 2010)	Creates and maintains direct traceability links between functional concerns and their respective implementations in code.
ThreeVaMar	(Rubin and Chechik 2010)	This algorithm accepts as input a model with duplications that represent systems and produces the model of a product line.
Feature Model Extraction	(Haslinger et al. 2011)	An algorithm reverse engineers a basic feature model from the feature sets, which describes the features each system provides.
RecFeat	(Nunes et al. 2012)	A history-sensitive heuristic for recovering features in code of degenerate program families.
ETHOM	(Segura et al. 2012)	Uses an evolutionary algorithm for the automated generation of feature models.
Clone-Differentiator Tool	(Xue 2012)	Automatically characterizes clones returned by a clone detector by differentiating Program Dependence Graphs (PDGs) of clones. It is able to provide a precise characterization of semantic differences of clones.
MapHist Tool	(Nunes et al. 2013)	MapHist tool applies heuristics to explore the evolution history of the family members in order to expand feature mappings in evolving program families.
SPLevo tools	(Klatt et al. 2013)	SPLevo is a software development tool supporting the consolidation of customized product copies into a Software Product Line.
Theme/SPL	(Araújo et al. 2013)	A tool to enhance feature modelling with traceability and improved support for cross-cutting concerns.
BUT4Reuse	(Martinez et al. 2014; Martinez et al. 2015)	This tool provides technologies for leveraging commonality and variability of software artefacts.
ExtractorPL	(Ziadi et al. 2014)	A language-independent approach which provides a quick automatic front-end to refactor a set of systems into an SPL.

Table 10 (continued)

Name	Reference	Description
ECCO Tool	(Fischer et al. 2014; 2015)	This tool automatically locate reusable parts in existing systems and compose a new system from a selection of desired features.
Model Driven SaaS	(Mohamed et al. 2014)	This eclipse plugin executes the QVT transformations that were defined based on the evolution rules.
AUFM Suite	(Bagheri et al. 2012)	In-house Eclipse-based plug-in for feature modeling.
JfeTkit	(Tang and Leung 2015)	JFeTkit (Java Feature Mining Toolkit) extracts featured code from the software legacy.
FMr-T	(Maâzoun et al. 2014b)	FMr-T (Feature Model recovery Tool) is a feature model extraction tool that identifies code variability.
ArborCraft	(Weston et al. 2009)	This tool suite automatically processes natural-language requirements documents into a candidate feature model.

and 2007. We observe a "boom" between 2011 and 2013, when 54.6 % of the papers were published.

4.8 RQ8 - Trends and Research Opportunities

During the analysis of the primary sources we identified some research gaps and limitations. In this section, we report the research opportunities and trends uncovered by our mapping study.

4.8.1 Automation and Tool Support

We observed that further studies should envisage the implementation of tools to automate the entire process of reengineering of existing variants to an SPL. In many papers the authors expose only an intention to provide a tool support to their methods. The first reason to provide tool support to the reengineering process is to reduce the manual effort (Stoermer and O'Brien 2001; Yang et al. 2009; Mefteh et al. 2014; Abbasi et al. 2014). Moreover, an automated process can improve the overall quality of the reengineering process, since this process is a labour-intensive task and error-prone (Stoermer and O'Brien 2001). In this sense, authors argue for the necessity of providing tool support, such as Sampath (2013), Passos et al. (2013), Zhang et al. (2011), de Oliveira et al. (2012), and Acher et al. (2012), enabling an easier and better application of their approaches.

Despite the need to automate the entire reengineering process, authors point out the missing tool support for specific tasks. For instance, for detection phase, Santos et al. point out as future research the use of test-based feature location to automate the mapping of features to source code (Santos et al. 2013). As another example, now for the analysis phase, Xue et al. (2010) mention the possible use of tools to automate reconciliation of inconsistent

Table 11 Tools per Phase and Strategy

Tool	Phases			Strategies					Input				Output			
	Detec.	Analy.	Trans.	Exp.	Stat.	Dyn.	IR	SB	Dom.	Requi.	Design	Code	Mapp.	Disc.	Repo.	Refac.
Variability to Aspect tool	✓	✓			✓							✓				✓
FeatureMapper	✓	✓		✓	✓						✓	✓	✓			
CoDEx Tool	✓	✓			✓					✓		✓	✓			
ThreeVaMar	✓	✓	✓		✓						✓			✓		
Feature Model Extraction	✓	✓			✓						✓			✓		
RecFeat	✓	✓	✓		✓					✓		✓				✓
ETHOM		✓						✓			✓			✓		
Clone-Differentiator Tool	✓	✓	✓				✓			✓	✓	✓				✓
MapHist Tool	✓	✓			✓					✓			✓			
SPLevo tools	✓	✓	✓		✓	✓						✓				✓
Theme/SPL	✓	✓	✓		✓					✓				✓		
BUT4Reuse	✓	✓	✓	✓	✓					✓	✓	✓	✓	✓	✓	✓
ExtractorPL	✓	✓					✓					✓	✓			
ECCO Tool	✓	✓	✓	✓	✓							✓	✓			✓
Model Driven SaaS			✓	✓	✓						✓	✓				✓
AUFM Suite	✓	✓		✓			✓		✓						✓	
JfeTkit	✓	✓	✓		✓							✓				✓
FMr-T	✓	✓					✓					✓		✓		
ArborCraft	✓	✓		✓	✓					✓				✓		

Fig. 9 Percentage of publications per type

feature models and Li et al. (2005) argue that there is still no tool for feature aggregation and abstraction. Regarding the transformation phase, Olszak and Jørgensen point out the labour-intensive task of manually annotating feature entry points (Olszak and Jørgensen 2012). She et al. describe as challenge the automation of feature location and dependency mining when the focus of reengineering is large-scale systems (She et al. 2011).

For those papers that provide a tool, the authors recommend possible improvements. Bécan (2013) and Merschen et al. (2011) point out that it is necessary to improve their tools. According to the authors, the usability has impact on the effort saved. With respect to the motivation to use their tools, Merschen et al. (2011), Damaševičius et al. (2012), Ferrari et al. (2013), and Acher et al. (2013) mention their intention on integrating their tools into standard frameworks and environments to make them useful for developers and engineers. So, besides the importance of existence of tools to support the reengineering, their usability and integration into popular development frameworks should be considered. Martinez et al. present a tool support for the reengineering process, however they argue for extend their tool to deal with different types of artefacts (Martinez et al. 2015).

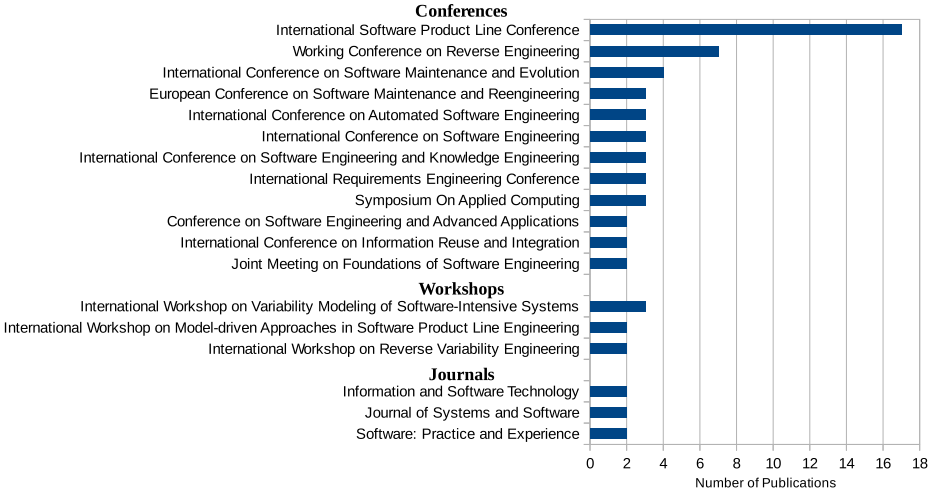


Fig. 10 Conferences, workshops and journals with most publications

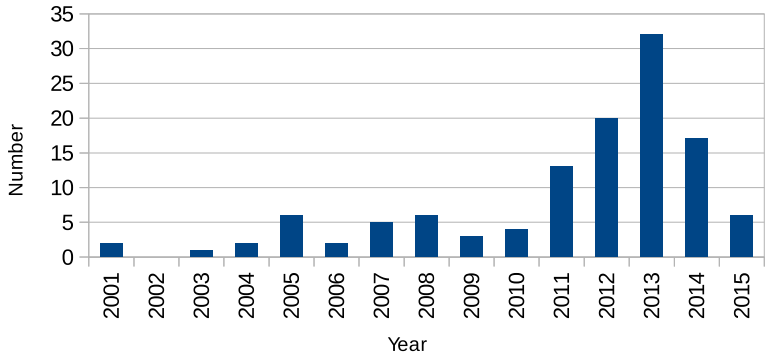


Fig. 11 Number of publications per year

In summary, given the increasing interest in SPLs, the implementation of tools to support the phases of the entire process is fundamental to the practice and use in the industry.

4.8.2 Exploiting Multiple Sources of Information for Reengineering

Another research direction observed is exploiting different information sources during the reengineering process. A research opportunity presented by Knodel et al. is using test cases, commonly available in most projects, in conjunction with other sources to determine features (Knodel et al. 2005). Trifu argues for the extraction of direct flow relations from sources other than the source code (Trifu 2010). Kelly et al. suggest exploring source code comments and documentation to enrich their approach for concept mining (Kelly et al. 2011). Eyal-Salman et al. indicate as future work the use of relationships between source code elements to improve the traceability and feature identification (Eyal Salman et al. 2013). Duszynski et al. (2011) and Peng et al. (2013) mentioned as research direction the use of design knowledge such as architecture models to allow the reengineering at a high abstraction level. Bécan et al. do not point out what specific source should be explored, but recommend that all artefacts that may be present in software projects can be used (Bécan et al. 2013). In the same way, Yu et al. envisage the use of multi-grained resources, such as code bases, historical code changes, mailing lists, bug databases, software descriptions, user evaluations, etc Yu et al. (2013). To have a benefit in using different sources of information, Kulesza et al. say that links between the different artefacts should be constantly managed (Kulesza et al. 2007). This enables the use of different sources in conjunction, such as proposed by Almeida et al. (2006), that recommend as future work the use of both domain analysis and domain design in the software evolution. In the same way She suggests the combination of bottom-up vs. top-down synthesis using artefacts at different levels of abstraction to cover different points of view (She 2013).

4.8.3 Feature Management

Feature management is an important task in the reengineering process, responsible for providing the variability among the features that compose the product variants.

An identified trend is the automated recovery of feature dependencies and interactions considering aspects such as non-functional characteristic of systems (Li et al. 2005; Ali et al. 2011; Bagheri et al. 2012). These aspects may help to refine the feature mappings

and improve the resulting SPL (Li et al. 2007). Many studies generate feature models as output but, in general, constraints, such as one feature requires or excludes another feature, are not considered (Haslinger et al. 2011; Eyal-Salman et al. 2012; Damaševičius et al. 2012; AL-Msie'deen et al. 2012; Al-msie'deen et al. 2013; Ferrari et al. 2013; Xue 2012; Ziadi et al. 2014; She et al. 2014). Automatic recovery of constraints is an open issue to be addressed in new studies. Some authors pointed out the research opportunity related with the support to introduce, move, or delete features on a migrated SPL (Polzer et al. 2012; de Oliveira et al. 2012). In fact it is not directly related with the reengineering process, but if considered during the reengineering, the future evolution and maintenance may be easier. Another research direction is the reengineering of partial variability, where a subset of features with variability are considered more important and hence given priority in the reengineering process, i.e. they will be migrated first, ahead of other lower priority features (Romero et al. 2013; Losavio et al. 2013).

4.8.4 Hybrid Approaches

Hybrid approaches can improve the results when compared with the application of only one type of strategy. For example, new approaches could consider the combination of different strategies. Dynamic analysis can be combined with static analysis such as recommended in the papers (Eisenbarth et al. 2001; Frenzel et al. 2007) or static analysis combined with information retrieval (Romero et al. 2013). Some future directions are related to the use of incremental and interactive approaches including the expert engineers in the automatic process. It will be useful in situations with unsound or incomplete input to provide the required information to enable the automated process (Haslinger et al. 2011; Davril et al. 2013).

Another research direction is the combination of techniques to better explore the artefacts used in the reengineering. For example the combination of Formal Concept Analysis and Latent Semantic Indexing to further explore the requirements artefacts (Eyal-Salman et al. 2012; AL-Msie'deen et al. 2013). Furthermore, some authors expose the opportunity of using additional techniques (Rubin and Chechik 2012b; Anwikar et al. 2012; Acher et al. 2012). Recently, studies applying search-based algorithms have appeared. The Search-based strategy has been little explored in the area of SPLs (Lopez-Herrejon et al. 2015) and has the potential to exploit hybrid approaches (Harman et al. 2014). Based on this, Lopez-Herrejon et al. point out the use of search-based algorithms to address many variability management challenges (Lopez-Herrejon et al. 2015).

4.8.5 New Measures and Metrics

Measures and metrics are fundamental to support the reengineering process. However, we observe that more specific factors should be considered during the reengineering tasks. Some research opportunities are presented next.

Some authors indicated as way to improve the results of the reengineering the use of new similarity measures. Rubin et al. point out the lack of alternative methods for calculating graph similarity to deal with model variants (Rubin and Chechik 2012a). Nöbauer et al. mentioned the need of sophisticated similarity calculation method to identify commonalities in existing products (Nöbauer et al. 2014b). Eyal-Salman et al. identified the opportunity of new research on the combination of lexical similarity with structural similarity to achieve better results on the detection phase (Eyal-Samal et al. 2013a, b, c); Niu et al. mention the necessity of novel ways to compute requirements similarity (Niu et al. 2014).

Studies with sophisticated metrics to validate the costs and benefits of the reengineering process should be carried out (Rubin et al. 2012). Research in this direction, mainly in real scenarios, can help to evaluate the effort saved (Bécan et al. 2013). Besides, it is important a more rigorous measurement and reporting for quantifying business benefits (Rubin 2014). For example, to assess how is the evolution of the system after the reengineering (Bécan 2013). In general, the reengineering will provide benefits for the existing systems, but how about the cost of adding new systems or new specific features?

Other possibilities for further investigation are regarding the existing relationships among artefacts, mainly source code (e.g., method call, class inheritance, etc.) (Klatt et al. 2014; Eyal Salman et al. 2013). This requires studies on semantic similarity measure among elements (Nie et al. 2012). The combination of measures and metrics can minimize the influence of the input to reach good results in the reengineering of systems in different domains (Falessi et al. 2010).

4.8.6 More Robust Empirical Evaluation

The great majority of the proposed approaches need better evaluation. In some cases a better evaluation is required because only academic and illustrative systems were used to introduce the approaches, however, they acknowledge the importance of using real case studies (Heidenreich et al. 2008; Van Der Storm 2007; Anwikar et al. 2012; Rubin and Chechik 2012b; Araújo et al. 2013; Bécan 2013). It is also common the authors evaluate their approaches with product variants of existing SPLs, an example is the use of ArgoUML-SPL. Using existing SPLs in the evaluation makes possible to compare the reengineered SPL with the original one. But they mention, as future work, empirical evaluation considering industrial partners (Knodel et al. 2005; Noor et al. 2008; Knodel et al. 2005; Klatt et al. 2013; Abbasi et al. 2014). Evaluation of the approaches in different domains and with complex case studies are mentioned as important future work (Ramos and Penteado 2008; Ziadi et al. 2012; Acher et al. 2013; Shao et al. 2013; Nöbauer et al. 2014a). Other authors just mention the need of further evaluation (Knodel et al. 2005; Alves et al. 2007; Maia et al. 2008; Breivold et al. 2008; Trifu 2010; Ali et al. 2011; Acher et al. 2011; Seidl et al. 2012; Segura et al. 2012; Linsbauer et al. 2013; Santos et al. 2013; Passos et al. 2013; Davril et al. 2013; Linsbauer et al. 2014; Chen et al. 2005; Mefteh et al. 2014; Acher et al. 2012; Hariri et al. 2013; Guzman and Maalej 2014; Eriksson et al. 2005; Mohamed et al. 2014; Kumaki et al. 2012; 2012). Besides further studies related with the better evaluation, another common issue is the lack of a framework for comparing reengineering approaches (Yang et al. 2009; Xue et al. 2010; Rubin and Chechik 2010; Segura et al. 2012; Nunes et al. 2012; AL-Msie'deen et al. 2012; Lohar et al. 2013).

4.8.7 Other Issues and Challenges

In the following we describe some trends and opportunities mentioned in few papers that can be a starting point to new studies.

New refactoring techniques should be proposed. In the results we observed a lack of studies on the transformation phase. In two papers Rubin et al. point out the need of sophisticated techniques for refactoring of models variants to generate an SPL (Rubin and Chechik 2010; 2012a). Maâzoun et al. cited as future work the use of semantics in the refactoring of SPLs (Maâzoun et al. 2014b). Moreover, together with these techniques, it is important to devise testing tasks to check the impact on the quality of SPL refactorings (Kolb et al. 2006).

Some authors point out the importance of creating general guidelines covering the recent advances of the field. Otsuka et al. (2011), Nunes et al. (2012), and Ziadi et al. (2012) argue about the creation of guidelines to formalize the tasks of their proposed approaches. In a similar way, other authors believe that the guidelines can lead to more automated support (Stoermer and O'Brien 2001; Ramos and Penteado 2008; Martinez et al. 2015). Kang et al. point out the need for guidelines for evaluating product line assets (Kang et al. 2005).

She cites the importance of future studies to cope with unsound or incomplete input (She 2013). Both Rubin et al. (2013) and Bayer et al. (2004) presented operators to support the reengineering of existing systems. An open research area is extending and refining this catalogue of operators as well as dealing with incomplete or uncertain input information.

Nunes et al. (2013) and Trifu (2010) purpose new research to better explore techniques of seeding for mining of features in the source code. Polzer et al. point out the importance of proposing approaches of general purpose to support the reengineering of system variants in different domains (Polzer et al. 2012). Heidenreich et al. argue about new studies to create more elaborated visualization tools to support feature mapping (Heidenreich et al. 2008). Visualization is also pointed as a trend by Hariri et al. (2013) and by Guzman and Maalej (2014). A challenge exposed by Schulze et al. is reuse of regulations of functional safety besides the implementation artefacts (Schulze et al. 2013). Finally, the reengineering seems to be a good context to deep analysis of the cost-benefit of the systematic reuse, i.e. it is an opportunity for the application of Value-Based Software Engineering (Zhang et al. 2011).

5 Threats to Validity

The validity threats we faced are related to the systematic mapping process. A first threat is concerned with the research questions. To minimize this kind of threat, we had several discussions about the questions and goals of our search. We argue the research questions reflect the goals of our work.

A second threat is about the terms used in the search queries. To address this threat we composed three groups of terms that best represent our goals. Most of the terms used were extracted from related works (Laguna and Crespo 2013; Lopez-Herrejon et al. 2015).

A third threat is concerned with the databases used. We perform the search for primary sources on eight databases. The databases selected are well known and include the most relevant ones, also we considered more databases than the related systematic mapping (Laguna and Crespo 2013).

A fourth threat to validity is our classification scheme. We created a classification scheme to enable answering our research questions. The steps to compose the presented classification scheme were: (i) first we determined six dimensions related to the research questions, (ii) then we collected and documented all relevant information from the primary sources taking into account the dimensions and research questions, (iii) then we analysed what of the identified information are similar or common in different studies, and finally (iv) one category for each similar/common item was created in the correlated dimension. Other researchers may possibly obtain another scheme.

A fifth threat to validity concerns the data extraction using the classification scheme. During the creation of the classification scheme, the data extracted from the primary sources was documented in a text document. This document was used to extract the information and when necessary the studies were reread to clarify some doubt about the right category for the paper. Also, we had many meetings and discussions about the extraction of the data.

The last threat is related to a possible incorrect identification of research gaps and limitations of existing studies. To reduce this threat we collected each mentioned limitation and described future work for each primary source. After collecting this information from all studies, we analyzed the data in order to identify common gaps and limitations.

6 Related Work

The related study most similar to ours is the work of Laguna and Crespo (2013). They performed a systematic mapping study on SPL evolution, but they also consider the refactoring of existing SPLs, which is not our focus. In contrast from Laguna and Crespo's mapping study, our study has the following differences:

- *Addition of a broader set of search terms* : to conduct the search we considered a broader set of terms related to reengineering and SPLs, furthermore we include a new set of terms related to feature location (see Section 3.2);
- *Inclusion of a different set of research questions*: our systematic mapping has seven research questions to shed more light in the reengineering process. Laguna and Crespo presented four research questions covering coarse-grained aspects of the reengineering process and SPL refactoring, on the other hand we have focus on fine-grained aspects such as phases and techniques/methods. Besides of mapping the works regarding the approach, techniques and challenges of the reengineering process, as done by Laguna and Crespo, we also mapped the common phases, the artefacts considered as input and produced as output during the obtaining of the SPLs;
- *Different analysis of case studies used for evaluation*: we collect in the primary sources and presented the case studies used to validate the proposed approaches;
- *Publication venues and evolution*: our work also presents analysis about the common publication venues preferred by researchers and evolution of the publications along the years;
- *Extended classification scheme*: we adopt a different classification scheme using a more fine-grained categorization regarding the reengineering phases, techniques and methods, and the type of input and output artefacts;
- *Updated primary sources*: Laguna and Crespo's mapping study considered papers published until 2011, on the other hand our mapping includes papers published until 2015.

Fenske et al. construct a detailed taxonomy of SPL reengineering, giving different names to distinct activities and showing their relationships (Fenske et al. 2013). In summary, they considered migration, refactoring and mapping in the reengineering process. In contrast, in this paper we considered the reengineering phases and strategies. Furthermore, their effort was only concentrated on the classification of a corpus of available work, presenting what studies exist in SPLs reengineering without providing further details or analysis of them. The scope of their study is smaller than ours. They considered only three dimensions regarding the purpose of the reengineering, the technique used, and the number of software systems used as input for the process. They included works that also perform the reengineering from a single system to an SPL, which is not our focus.

Lozano presents a survey of approaches to detect variability concepts in source code (Lozano 2011). Her focus is specifically the detection of variabilities. One of the conclusions pointed by Lozano is that some techniques, e.g. to address architectural degradation, are limited to address SPLs from single products. This conclusion corroborates our motivation

in performing this mapping study to map works starting from a set of products instead of a single product. Tiarks et al. present a state-of-art of clone detection with focus on migration (Tiarks et al. 2011). They performed an assessment of clone detection in source code. In contrast with those studies, we focus our study on the entire reengineering process, not only presenting the scenario of detection phase.

Two systematic literature reviews have as focus the requirements engineering for software product lines. Alves et al. have used SPLE adoption strategies to classify the papers (Alves et al. 2010). From the results we observed that 39 % of the papers employed an extractive approach, on the other hand 57 % of the papers adopt a proactive approach. The authors suggest to researchers and practitioners focus more on extractive strategy, as well as reactive. The work of Bakar et al. presents approaches that only extract features from natural language requirements for reuse in SPLE (Bakar et al. 2015). This study is related to ours, however, it reports only approaches that use requirements as input. Differently, our study also considers the other artefacts used in the reengineering.

Koziolek et al. reported an exploratory case study on experiences and lessons learned from domain analysis in four large-scale cases on more than 20 industrial software systems (Koziolek et al. 2015). After the domain analysis only one case study resulted in an SPL. For the other cases stakeholders decided to use smaller integration scenarios, mainly due to high migration costs in the industrial domain and because of the business flexibility. According to the authors, business flexibility is often an argument against an SPL approach once some stakeholders were afraid of tightly collaborating with other business units. This study shows that the reengineering is not the best choice in such conditions. However, the authors point that more studies are necessary to further support their findings.

Harman et al. present a survey with directions for future work on Search-Based Software Engineering (SBSE) to SPLs (Harman et al. 2014). Regarding the reengineering process, the authors present some studies on the reverse engineer of feature models from a set of instances applying SBSE techniques. Another claim made by the authors is that the recent advances in genetic improvement might be exploited by SPL researchers and practitioners. We also have presented an overview and roadmap on the use of genetic improvement to SPLs (Lopez-Herrejon et al. 2015). Some connections are drawn between recent and ongoing research on reverse engineering SPLs and their evolution with the GISMOE approach.

Galster et al. performed a literature review about variability in software systems (Galster et al. 2014). As result the authors proposed an empirically grounded classification of the dimensions of variability. They also attest lack of evidence for the validity of existing approaches. Chen and Babar present a systematic review on the field of variability management (Chen and Babar 2011). The authors pointed out that there is a lack of robust evaluation for the approaches. Nevertheless, they also show that most of the studies report positive effects of the proposed variability management approaches. About variability management, Metzger and Pohl presented achievements and challenges of the field (Metzger and Pohl 2014). The authors identified some research challenges that existed for quite some time are still opened. For instance, product line quality assurance techniques, scoping, domain design, application requirements engineering, as well as application design and realization. Despite of the three works have done an extensive survey of variability and variability management literature, none of them discussed about reengineering of existing system.

The study of Heradio et al. reported a bibliometric analysis of research on software product lines (Heradio et al. 2016). The goal of their paper is to cover the entire field of SPLs. They identify the most influential publications, the most researched topics, and how the interest in SPL topics has evolved along the time. They conclude that software

architecture was the initial motor of research in SPLs, and that feature modeling has been the most important topic for the last fifteen years. Differently of this bibliometric analysis, we present here a systematic mapping with focus in the sub-field of reengineering.

7 Concluding Remarks

In this paper we describe the results of a systematic mapping study on the reengineering of existing systems to an SPL. We observed that studies in this field are presented in a wide range of venues such as conferences, workshops, journals, technical reports, PhD and master theses, and book chapters. The increase in the number of publications until 2013 points out a great interest in this topic. Despite of a decrease in the number of publications in 2014-2015, the topic of reenginering still has attention of the research community. Different strategies based on existing methods present in Software Engineering are used to support the reengineering process. Static analysis, Expert-driven, and Information Retrieval are the most common strategies applied. Dynamic analysis has few works along the years and the Search-based strategy has appeared recently. Artefacts of almost all software engineering process are considered as input and output. Source code and Requirements are the most common inputs. Feature discovery and Source code refactored are the most common outputs. To evaluate the proposed approaches, several systems are used. In the category of industrial systems we collected 50 different systems. ArgoUML and Linux Kernel are the most common. In the category of academic systems, we collected 32 systems used in evaluation. MobileMedia is the most common. Regarding tools supporting the reengineering process, we identified 15 tools in the primary sources.

During the classification and reading of the studies, we could observe the existence of research opportunities and trends. So, eight major areas for future research are presented, namely: automation and tool support, exploiting multiple sources of information for reengineering, feature management, hybrid approaches, refactoring techniques, need of use guidelines, new measures and metrics, and more robust empirical evaluation.

We hope that this mapping not only motivate new research on this topic, but also encourages software companies to consider the implementation of the systematic reuse of their products. C&O is a common practice in industry, so companies have available the resources needed for the reengineering. From the findings presented in this mapping study companies can be aware of the reengineering process to obtain an SPL, the available tools, artifacts commonly used and created, as well as, approaches proposed to perform the reengineering. As a result of the systematic reuse, companies are able to maintain existing products and evolve their portfolio of products by reusing existing artifacts in an easier way.

Acknowledgments This work was supported by the Brazilian Agencies CAPES: 007126/2014-00 and CNPq: 453678/2014-9 and 305358/2012-0, and Austrian Science Fund (FWF): P 25289-N15.

References

Alves V, Niu N, Alves C, Valença G. (2010) Requirements engineering for software product lines: A systematic literature review. Inf Softw Technol 52(8):806–820. doi:10.1016/j.infsof.2010.03.014

Assunção WKG, Vergilio SR (2014) Feature location for software product line migration: A mapping study 18th Software Product Line Conference - 2nd International Workshop on REverse Variability Engineering (REVE), pp 1–8. doi:10.1145/2647908.2655967

Bachmann F, Clements P (2005) Variability in software product lines. Tech. Rep. CMU/SEI-2005-TR-012, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA

Bakar NH, Kasirun ZM, Salleh N (2015) Feature extraction approaches from natural language requirements for reuse in software product lines: A systematic literature review. J Syst Softw 106:132–149. doi:10.1016/j.jss.2015.05.006

Chen L, Babar MA (2010) 14th International Conference Software Product Lines: Going Beyond (SPLC 2010), chap. Variability Management in Software Product Lines: An Investigation of Contemporary Industrial Challenges. Springer Berlin Heidelberg, Berlin, pp 166–180. doi:10.1007/978-3-642-15579-6_12

Chen L, Babar MA (2011) A systematic review of evaluation of variability management approaches in software product lines. Inf Softw Technol 53(4):344–362. doi:10.1016/j.infsof.2010.12.006

Chikofsky E, Cross J.HI (1990) Reverse engineering and design recovery: a taxonomy. IEEE Softw 7(1):13–17. doi:10.1109/52.43044

Clements P, Northrop L (2001) Software Product Lines: Practices and Patterns. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA

Cornelissen B, Zaidman A, van Deursen A, Moonen L, Koschke R (2009) A systematic survey of program comprehension through dynamic analysis. IEEE Trans Softw Eng 35(5):684–702. doi:10.1109/TSE.2009.28

Demeyer S, Ducasse S, Nierstrasz O (2009) Object-oriented reengineering patterns. Square Bracket associates, Switzerland. Version of 2009-09-28

Dit B, Revelle M, Gethers M, Poshyvanyk D (2013) Feature location in source code: a taxonomy and survey. Journal of Software: Evolution and Process 25(1):53–95. doi:10.1002/smr.567

Dubinsky Y, Rubin J, Berger T, Duszynski S, Becker M, Czarnecki K (2013) An exploratory study of cloning in industrial software product lines 17th European Conference on Software Maintenance and Reengineering (CSMR), pp 25–34. doi:10.1109/CSMR.2013.13

Faust D, Verhoef C (2003) Software product line migration and deployment. Software: Practice and Experience 33(10):933–955

Fenske W, Thüm T, Saake G (2013) A taxonomy of software product line reengineering 8th International Workshop on Variability Modelling of Software-Intensive Systems, VaMoS 2014, pp 1–8. ACM, New York, NY, USA. doi:10.1145/2556624.2556643

Galster M, Weyns D, Tofan D, Michalik B, Avgeriou P (2014) Variability in software systems - systematic literature review. IEEE Trans Softw Eng 40(3):282–306. doi:10.1109/TSE.2013.56

Harman M, Jia Y, Krinke J, Langdon WB, Petke J, Zhang Y (2014) Search based software engineering for software product line engineering: A survey and directions for future work 18th International Software Product Line Conference - Volume 1, SPLC 2014, pp 5–18. ACM, New York, NY, USA. doi:10.1145/2648511.2648513

Harman M, Mansouri SA, Zhang Y (2009) Search based software engineering: A comprehensive analysis and review of trends techniques and applications. Tech. Rep. Technical Report TR-09-03, Department of Computer Science, King's College London

Heradio R, Perez-Morago H, Fernandez-Amoros D, Cabrerizo FJ, Herrera-Viedma E (2016) A bibliometric analysis of 20 years of research on software product lines. Inf Softw Technol 72:1–15. doi:10.1016/j.infsof.2015.11.004

Kang K, Cohen S, Hess J, Novak W, Peterson A (1990) Feature-Oriented Domain Analysis (FODA) Feasibility Study. Tech. Rep. CMU/SEI-90-TR-21, SEI, CMU

Koziolek H, Goldschmidt T, Gooijer T, Domis D, Sehestedt S, Gamer T, Aleksy M (2015) Assessing software product line potential: an exploratory industrial case study. doi:10.1007/s10664-014-9358-0

Krueger CW (1992) Software reuse. ACM Comput Surv (CSUR) 24(2):131–183. doi:10.1145/130844.130856

Krueger CW (2002) Easing the transition to software mass customization Software Product-Family Engineering, pp 282–293. Springer

Laguna MA, Crespo Y (2013) A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring. Sci Comput Program 78(8):1010–1034. doi:10.1016/j.scico.2012.05.003

Linden FJVD, Schmid K, Rommes E (2007) Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering. Springer-Verlag New York, Inc., Secaucus, NJ, USA

Lopez-Herrejon R, Linsbauer L, Assunção W. K, Fischer S, Vergilio SR, Egyed A (2015) Genetic improvement for software product lines: An overview and a roadmap 2015 Annual Conference on Genetic and

Evolutionary Computation, Genetic Improvement 2015 Workshop, GECCO, pp 823–830. ACM, New York, NY, USA. doi:10.1145/2739482.2768422
Lopez-Herrejon R, Linsbauer L, Egyed A (2015) A systematic mapping study of search-based software engineering for software product lines. Inf Softw Technol 61(0):33–51. doi:10.1016/j.infsof.2015.01.008
Lozano A (2011) An overview of techniques for detecting software variability concepts in source code Workshops - Advances in Conceptual Modeling: Recent Developments and New Directions, LNCS, vol. 6999, pp 141–150. Springer Berlin Heidelberg. doi:10.1007/978-3-642-24574-9_19
Manning CD, Raghavan P, Schütze H., et al. (2008) Introduction to information retrieval, vol 1, Cambridge University Press
Metzger A, Pohl K (2014) Software product line engineering and variability management: Achievements and challenges Future of Software Engineering, FOSE 2014, pp 70–84. ACM, New York, NY, USA. doi:10.1145/2593882.2593888
Petersen K, Feldt R, Mujtaba S, Mattsson M (2008) Systematic mapping studies in software engineering. British Computer Society, Swinton, UK, pp 68–77
Petersen K, Vakkalanka S, Kuzniarz L (2015) Guidelines for conducting systematic mapping studies in software engineering: An update. Inf Softw Technol 64:1–18. doi:10.1016/j.infsof.2015.03.007
Pohl K, Böckle G. (2005) Linden, F.J.v.d.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, Inc., Secaucus, NJ, USA
Riva C, Del Rosso C (2003) Experiences with software product family evolution Sixth International Workshop on Principles of Software Evolution (IWPSE), pp 161–169. doi:10.1109/IWPSE.2003.1231223
Rubin J, Chechik M (2013) A survey of feature location techniques. In: Reinhartz-Berger I., Sturm A., Clark T., Cohen S., Bettin J. (eds) Domain Engineering, pp 29–58. Springer Berlin Heidelberg. doi:10.1007/978-3-642-36654-3_2
Svahnberg M, van Gurp J, Bosch J (2005) A taxonomy of variability realization techniques: Research articles. Software - Practice and Experience 35(8):705–754. doi:10.1002/spe.v35:8
Tiarks R, Koschke R, Falke R (2011) An extended assessment of type-3 clones as detected by state-of-the-art tools. Softw Qual J 19(2):295–331. doi:10.1007/s11219-010-9115-6
Wagner C (2014) Model-Driven Software Migration: A Methodology Reengineering, Recovery and Modernization of Legacy Systems, Springer Vieweg
Wichmann BA, Canning AA, Clutterbuck DL, Winsborrow LA, Ward NJ, Marsh DWR (1995) Industrial perspective on static analysis. Softw Eng J 10(2):69–75
Wohlin C (2014) Guidelines for snowballing in systematic literature studies and a replication in software engineering 18th International Conference on Evaluation and Assessment in Software Engineering, EASE'14, pp 38:1–38:10. ACM, New York, NY, USA. doi:10.1145/2601248.2601268

Primary Sources

Abbasi E, Acher M, Heymans P, Cleve A (2014) Reverse engineering web configurators IEEE Conference on software maintenance, reengineering and reverse engineering (CSMR-WCRE), pp 264–273. doi:10.1109/CSMR-WCRE.2014.6747178
Acher M, Cleve A, Collet P, Merle P, Duchien L, Lahire P (2011) Reverse engineering architectural feature models. In: Crnkovic I, Gruhn V, Book M (eds) Software Architecture, Lecture Notes in Computer Science, vol. 6903, pp 220–235. Springer Berlin Heidelberg. doi:10.1007/978-3-642-23798-0_25
Acher M, Cleve A, Collet P, Merle P, Duchien L, Lahire P (2013) Extraction and evolution of architectural variability models in plugin-based systems. doi:10.1007/s10270-013-0364-2
Acher M, Cleve A, Perrouin G, Heymans P, Vanbeneden C, Collet P, Lahire P (2012) On extracting feature models from product descriptions. In: 6th international workshop on variability modeling of software-intensive systems, vamos. ACM, New York, pp 45–54. doi:10.1145/2110147.2110153
AL-Msie'deen R, Seriai A, Huchard M, Urtado C, Vauttier S, Salman H (2013) Feature location in a collection of software product variants using formal concept analysis. In: Favaro J, Morisio M (eds) Safe and Secure Software Reuse, Lecture Notes in Computer Science, vol. 7925, pp 302–307. Springer Berlin Heidelberg. doi:10.1007/978-3-642-38977-1_22
Al-msie'deen R, Seriai A, Huchard M, Urtado C, Vauttier S (2013) Mining features from the object-oriented source code of software variants by combining lexical and structural similarity. In: IEEE 14th international conference on information reuse and integration (IRI), pp 586–593. doi:10.1109/IRI.2013.6642522

AL-Msie'deen R, Seriai A, Huchard M, Urtado C, Vauttier S, Salman H (2012) An approach to recover feature models from object-oriented source code. Journé,e Lignes de Produits pp 15–26

Ali N, Wu W, Antoniol G, Di Penta M, Guéhéneuc Y, Hayes J (2011) Moms: Multi-objective miniaturization of software. In: 27th IEEE international conference on software maintenance (ICSM), pp 153–162

Almeida E, Mascena J, Cavalcanti A, Alvaro A, Garcia V, Lemos Meira S, Lucrédio D (2006) The domain analysis concept revisited: A practical approach. In: Morisio M (ed) Reuse of Off-the-Shelf Components, Lecture Notes in Computer Science, vol. 4039, pp. 43–57. Springer Berlin Heidelberg. doi:10.1007/11763864_4

Alves V, Matos Pedro J, Cole L, Vasconcelos A, Borba P, Ramalho G (2007) Extracting and evolving code in product lines with aspect-oriented programming. In: Rashid A, Aksit M (eds) Transactions on Aspect-Oriented Software Development IV, Lecture Notes in Computer Science, vol. 4640, pp 117–142 Springer Berlin Heidelberg. doi:10.1007/978-3-540-77042-8_5

Alves V, Schwanninger C, Barbosa L, Rashid A, Sawyer P, Rayson P, Pohl C, Rummler A (2008) An exploratory study of information retrieval techniques in domain analysis. In: 12th international software product line conference, SPLC, pp 67–76. doi:10.1109/SPLC.2008.18

Anwikar V, Naik R, Contractor A, Makkapati H (2012) Domain-driven technique for functionality identification in source code. SIGSOFT Softw Eng Notes 37(3):1–8. doi:10.1145/180921.2180923

Araújo JA, Goulão M, Moreira A, Simão I, Amaral V, Baniassad E (2013) Advanced modularity for building SPL feature models: a model-driven approach. In: 28th symposium on applied computing, SAC. ACM, New York, 1246–1253. doi:10.1145/2480362.2480596

Bagheri E, Ensan F, Gasevic D (2012) Decision support for the software product line domain engineering lifecycle. Int Conf Autom Softw Eng 19(3):335–377. doi:10.1007/s10515-011-0099-7

Bayer J, Forster T, Ganesan D, Girard JF, John I, Knodel J, Kolb R, Muthig D (2004) Definition of reference architectures based on existing systems. Tech. Rep. Report No. 034.04/E, Fraunhofer IESE-report No 034.04/E

Bécan G. (2013) Reverse engineering feature models in the real. Tech. Rep. dumas-00855005 Centre National de la Recherche Scientifique

Bécan G, Acher M, Baudry B, Ben Nasr S (2013) Breathing ontological knowledge into feature model management. Tech. Rep. RT-0441, INRIA - Institut National des Sciences Appliqué,es - Université de Rennes 1

Boutkova E, Houdek F (2011) Semi-automatic identification of features in requirement specifications. In: 19th IEEE international requirements engineering conference (RE), pp 313–318. doi:10.1109/RE.2011.6051627

Breivold H, Larsson S, Land R (2008) Migrating Industrial tab9 towards Software Product Lines: Experiences and Observations through Case Studies. In: 34th euromicro conference on software engineering and advanced applications (SEAA), pp 232–239. doi:10.1109/SEAA.2008.13

Chen K, Zhang W, Zhao H, Mei H (2005) An approach to constructing feature models based on requirements clustering. In: 13th IEEE international requirements engineering conference (RE), pp 31–40. doi:10.1109/RE.2005.9

Damaševičius R, Paškevičius P, Karčiauskas E, Marcinkevičius R (2012) Automatic extraction of features and generation of feature models from java programs. Inform Technol Control 41(4): 376–384

Davril JM, Delfosse E, Hariri N, Acher M, Cleland-Huang J, Heymans P (2013) Feature model extraction from large collections of informal product descriptions. In: 9th joint meeting on foundations of software engineering, ESEC/FSE. ACM, New York, pp 290–300. doi:10.1145/2491411.2491455

Duszynski S, Knodel J, Becker M (2011) Analyzing the source code of multiple software variants for reuse potential. In: 18th working conference on reverse engineering (WCRE), pp 303–307. doi:10.1109/WCRE.2011.44

Eisenbarth T, Koschke R, Simon D (2001) Derivation of feature component maps by means of concept analysis. In: European conference on software maintenance and reengineering (CSMR), pp 176–179. doi:10.1109/.2001.914982

Eriksson M, Morast H, Börstler J, Borg K (2005) The pluss toolkit?: Extending telelogic doors and ibm-rational rose to support product line use case modeling. In: 20th IEEE/ACM international conference on automated software engineering, ASE. ACM, New York, pp 300–304. doi:10.1145/1101908.1101955

Eyal Salman H, Djamel Seriai A, Dony C, Al-Msie'Deen R (2013) Identifying traceability links between product variants and their features. In: 1st international workshop on reverse variability engineering (REVE). Genova, Italie, pp 17–22

Eyal-Salman H, Seriai A, Dony C (2014) Feature location in a collection of product variants: Combining information retrieval and hierarchical clustering. In: 26th international conference on software engineering and knowledge engineering (SEKE)

Eyal-Salman H, Seriai A, Dony C (2013a) Feature-to-code traceability in a collection of product variants using formal concept analysis and information retrieval. In: 39th euromicro conference on software engineering and advanced applications (SEAA), pp 1–8

Eyal-Salman H, Seriai A, Dony C (2013b) Feature-to-code traceability in a collection of software variants: Combining formal concept analysis and information retrieval. In: IEEE 14th international conference on information reuse and integration (IRI), pp 209–216. doi:10.1109/IRI.2013.6642474

Eyal-Salman H, Seriai A, Dony C (2013c) Feature-to-code Traceability in Legacy Software Variants. In: 39th euromicro conference on software engineering and advanced applications (SEAA), pp 57–61. doi:10.1109/SEAA.2013.65

Eyal-Salman H, Seriai A, Dony C, Al-msie'deen R (2012) Recovering traceability links between feature models and source code of product variants. In: VARIability for you workshop: Variability modeling made useful for everyone, VARY. ACM, New York, pp 21–25. doi:10.1145/2425415.2425420

Falessi D, Cantone G, Canfora G (2010) A comprehensive characterization of NLP techniques for identifying equivalent requirements. In: 2010 ACM-IEEE International symposium on empirical software engineering and measurement, ESEM. ACM, New York, pp 1–10. doi:10.1145/1852786.1852810

Faust D, Verhoef C (2003) Software product line migration and deployment. Softw: Pract Experience 33(10):933–955

Ferrari A, Spagnolo GO, Dell'Orletta F (2013) Mining commonalities and variabilities from natural language documents. In: 17th international software product line conference, SPLC. ACM, New York, pp 116–120. doi:10.1145/2491627.2491634

Fischer S, Linsbauer L, Lopez-Herrejon R, Egyed A (2015) The ECCO tool: Extraction and composition for clone-and-own. In: IEEE/ACM 37th IEEE international conference on software engineering (ICSE), vol. 2, pp 665–668. doi:10.1109/ICSE.2015.218

Fischer S, Linsbauer L, Lopez-Herrejon R, Egyed A (2014) Enhancing clone-and-own with systematic reuse for developing software variants. In: IEEE 30th international conference on software maintenance and evolution, ICSME. IEEE Computer Society, Washington, DC, pp 391–400. doi:10.1109/ICSME.2014.61

Frenzel P, Koschke R, Breu A, Angstmann K (2007) Extending the reflexion method for consolidating software variants into product lines. In: 14th working conference on reverse engineering (WCRE), pp 160–169. doi:10.1109/WCRE.2007.28

Gamez N, Fuentes L (2011) Software product line evolution with cardinality-based feature models. In: Schmid K (ed) Top Productivity through Software Reuse, Lecture Notes in Computer Science, vol. 6727, pp 102–118. Springer Berlin Heidelberg. doi:10.1007/978-3-642-21347-2_9

Gamez N, Fuentes L (2013) Architectural evolution of famiware using cardinality-based feature models. Inf Softw Technol 55(3):563–580. doi:10.1016/j.infsof.2012.06.012. Special Issue on Software Reuse and Product Lines

Gharsellaoui H, Maazoun J, Bouassida N, Ben Ahmed S, Ben-Abdallah H (2015) A real-time scheduling of reconfigurable os tasks with a bottom-up spl design approach. In: 2015 International conference on evaluation of novel approaches to software engineering (ENASE), pp 169176

Guzman E, Maalej W (2014) How do users like this feature? a fine grained sentiment analysis of app reviews. In: 22nd IEEE international requirements engineering conference (RE), pp 153–162. doi:10.1109/RE.2014.6912257

Hariri N, Castro-Herrera C, Mirakhorli M, Cleland-Huang J, Mobasher B (2013) Supporting domain analysis through mining and recommending features from online product listings. IEEE Trans Softw Eng 39(12):1736–1752. doi:10.1109/TSE.2013.39

Haslinger E, Lopez-Herrejon R, Egyed A (2011) Reverse engineering feature models from programs' feature sets. In: 18th working conference on reverse engineering (WCRE), pp 308–312. doi:10.1109/WCRE.2011.45

Heidenreich F, Kopcsek J, Wende C (2008) Featuremapper: Mapping features to models. Companion of the 30th international conference on software engineering, ICSE companion. ACM, New York, pp 943–944. doi:10.1145/1370175.1370199

Kang K, Kim M, Lee J, Kim B (2005) Feature-oriented re-engineering of legacy systems into product line assets - a case study. In: Obbink H, Pohl K (eds) Software Product Lines, Lecture Notes in Computer Science, vol. 3714, pp 45–56. Springer Berlin Heidelberg. doi:10.1007/11554844_6

Kelly M, Alexander J, Adams B, Hassan A (2011) Recovering a balanced overview of topics in a software domain. In: 11th IEEE international working conference on source code analysis and manipulation (SCAM), pp 135–144. doi:10.1109/SCAM.2011.23x

Klatt B, Krogmann K, Seidl C (2014) Program dependency analysis for consolidating customized product copies. In: 30th IEEE international conference on software maintenance and evolution (ICSME), pp 496–500

Klatt B, Küster M, Krogmann K (2013) A graph-based analysis concept to derive a variation point design from product copies. In: International workshop on reverse variability engineering (REVE), pp 1–8

Knodel J, Forster T, Girard JF (2005) Comparing design alternatives from field-tested systems to support product line architecture design. In: Ninth european conference on software maintenance and reengineering (CSMR), pp 344–353. doi:10.1109/CSMR.2005.18

Knodel J, John I, Ganesan D, Pinzger M, Usero F, Arciniegas J, Riva C (2005) Asset recovery and their incorporation into product lines. In: 12th working conference on reverse engineering (WCRE), pp 1–10. doi:10.1109/WCRE.2005.8

Kolb R, Muthig D, Patzke T, Yamauchi K (2006) Refactoring a legacy component for reuse in a software product line: A case study. J Softw Maint Evol Res Pract 18(2):109–132. doi:10.1002/smr.v18:2

Koziolek H, Goldschmidt T, de Gooijer T, Domis D, Sehestedt S (2013) Experiences from identifying software reuse opportunities by domain analysis. In: 17th international software product line conference, SPLC. ACM, New York, pp 208–217. doi:10.1145/2491627.2491641

Kulesza U, Alves V, Garcia A, Neto A, Cirilo E, Lucena C, Borba P (2007) Mapping features to aspects: A model-based generative approach. In: Moreira A, Grundy J (eds) Early Aspects: Current Challenges and Future Directions, Lecture Notes in Computer Science, vol. 4765, pp 155–174. Springer Berlin Heidelberg. doi:10.1007/978-3-540-76811-1_9

Kumaki K, Tsuchiya R, Washizaki H, Fukazawa Y (2012) Supporting commonality and variability analysis of requirements and structural models. In: 16th international software product line conference - volume 2, SPLC. ACM, New York, pp 115–118. doi:10.1145/2364412.2364431

Lago P, Vliet H (2004) Observations from the recovery of a software product family. In: Nord R (ed) Software Product Lines, Lecture Notes in Computer Science, vol. 3154, pp 214–227. Springer Berlin Heidelberg. doi:10.1007/978-3-540-28630-1_13

Li S, Chen F, Liang Z, Yang H (2005) Using feature-oriented analysis to recover legacy software design for software evolution. In: International conference on software engineering and knowledge engineering (SEKE), pp 336–341

Li Y, Yin J, Shi D, Li Y, Dong J (2007) Software product line oriented feature map. In: Shi Y, Albada G, Dongarra J, Sloot P (eds) International Conference on Computational Science (ICCS), Lecture Notes in Computer Science, vol. 4488, pp 1115–1122. Springer Berlin Heidelberg. doi:10.1007/978-3-540-72586-2_156

Linsbauer L, Angerer F, Grünbacher P, Lettner D, Prähofer H, Lopez-Herrejon R, Egyed A (2014) Recovering feature-to-code mappings in mixed-variability software systems. In: IEEE 30th international conference on software maintenance and evolution, ICSME

Linsbauer L, Lopez-Herrejon ER, Egyed A (2013) Recovering traceability between features and code in product variants. In: 17th international software product line conference, SPLC. ACM, New York, pp 131–140. doi:10.1145/2491627.2491630

Linsbauer L, Lopez-Herrejon R, Egyed A (2014) Feature model synthesis with genetic programming. In: Le Goues C, Yoo S (eds) 6th Symposium on Search-Based Software Engineering (SSBSE), Lecture Notes in Computer Science, vol. 8636, pp 153–167. Springer International Publishing. doi:10.1007/978-3-319-09940-8_11

Lohar S, Amornborvornwong S, Zisman A, Cleland-Huang J (2013) Improving trace accuracy through data-driven configuration and composition of tracing features. In: 9th joint meeting on foundations of software engineering, ESEC/FSE. ACM, New York, pp 378–388. doi:10.1145/2491411.2491432

Lopez-Herrejon R, Linsbauer L, Galindo JA, Parejo J, Benavides D, Segura S, Egyed A (2015) An assessment of search-based techniques for reverse engineering feature models. J Syst Softw 103:353–369. doi:10.1016/j.jss.2014.10.037

Losavio F, Ordaz O, Levy N, Baiotto A (2013) Graph modelling of a refactoring process for product line architecture design. In: 39th latin american computing conference (CLEI), pp 1–12. doi:10.1109/CLEI.2013.6670632

Maazoun J, Bouassida N, Ben-Abdallah H (2014) A bottom up spl design method. In: 2nd international conference on model-driven engineering and software development (MODELSWARD), pp 309–316

Maâzoun J, Bouassida N, Ben-Abdallah H (2014) Feature model recovery from product variants based on a cloning technique. In: 26th international conference on software engineering and knowledge engineering (SEKE)

Maia MDA, Sobreira V, Paixão KR, Amo S, Silva IR (2008) Using a sequence alignment algorithm to identify specific and common code from execution traces. In: 4th international workshop on program comprehension through dynamic analysis (PCODA), pp 6–10

Martinez J, Ziadi T, Bissyandé TF, Klein J, Le Traon Y (2015) Bottom-up adoption of software product lines: a generic and extensible approach. In: 19th international conference on software product line, SPLC ACM, New York, pp 101–110. doi:10.1145/2791060.2791086

Martinez J, Ziadi T, Klein J, le Traon Y (2014) Identifying and visualising commonality and variability in model variants. In: Cabot J, Rubin J (eds) Modelling Foundations and Applications, Lecture Notes in Computer Science, vol. 8569, pp 117–131. Springer International Publishing. doi:10.1007/978-3-319-09195-2_8

Mefteh M, Bouassida N, Ben-Abdallah H (2014) Feature model extraction from documented UML use case diagrams. Ada User J 35(2):107–116

Merschen D, Polzer A, Botterweck G, Kowalewski S (2011) Experiences of applying model-based analysis to support the development of automotive software product lines. In: 5th international workshop on variability modelling of software-intensive systems, vamos. ACM, New York, pp 141–150. doi:10.1145/1944892.1944910

Mohamed F, Abu-Matar M, Mizouni R, Al-Qutayri M, Al Mahmoud Z (2014) Saas dynamic evolution based on model-driven software product lines. In: 6th international conference on cloud computing technology and science (cloudcom), pp 292–299. doi:10.1109/CloudCom.2014.131

Mu Y, Wang Y, Guo J (2009) Extracting software functional requirements from free text documents. In: International conference on information and multimedia technology (ICIMT), pp 194–198. doi:10.1109/ICIMT.2009.47

Nie K, Zhang L, Geng Z (2012) Product line variability modeling based on model difference and merge. In: IEEE 36th annual computer software and applications conference workshops (COMPSACW), pp 509–513. doi:10.1109/COMPSACW.2012.95

Niu N, Savolainen J, Niu Z, Jin M, Cheng JR (2014) A systems approach to product line requirements reuse. IEEE Syst J 8(3):827–836. doi:10.1109/JSYST.2013.2260092

Nöbauer M, Seyff N, Groher I (2014) Inferring variability from customized standard software products. In: 18th international software product line booktitle - volume 1, SPLC. ACM, New York, pp 284–293. doi:10.1145/2648511.2648544

Nöbauer M, Seyff N, Groher I (2014) Similarity analysis within product line scoping: An evaluation of a semi-automatic approach. In: Jarke M, Mylopoulos J, Quix C, Rolland C, Manolopoulos Y, Mouratidis H, Horkoff J(eds) Advanced Information Systems Engineering, Lecture Notes in Computer Science, vol. 8484, pp 165–179. Springer International Publishing. doi:10.1007/978-3-319-07881-6_12

Noor M, Grünbacher P, Hoyer C (2008) A collaborative method for reuse potential assessment in reengineering-based product line adoption. In: Meyer B, Nawrocki J, Walter B (eds) Balancing Agility and Formalism in Software Engineering, Lecture Notes in Computer Science, vol. 5082, pp 69–83. Springer Berlin Heidelberg. doi:10.1007/978-3-540-85279-7_6

Nunes C, Garcia A, Lucena C, Lee J (2012) History-sensitive heuristics for recovery of features in code of evolving program families. In: 16th international software product line conference - volume 1, SPLC. ACM, New York, pp 136–145. doi:10.1145/2362536.2362556

Nunes C, Garcia A, Lucena C, Lee J (2013) Heuristic expansion of feature mappings in evolving program families. Software: Practice and Experience pp 1–35. doi:10.1002/spe.2200

de Oliveira AL, Ferrari FC, Braga RT, Penteado RA, de Camargo VV (2012) Restructuring frameworks towards framework product lines. In: Latin american workshop on aspect-oriented software development (LA-WASP), pp 1–2

Olszak A, Jørgensen BN (2012) Remodularizing java programs for improved locality of feature implementations in source code. Sci Comput Program 77(3):131–151. doi:10.1016/j.scico.2010.10.007

Otsuka J, Kawarabata K, Iwasaki T, Uchiba M, Nakanishi T, Hisazumi K (2011) Small inexpensive core asset construction for large gainful product line development: Developing a communication system firmware product line. In: 15th international software product line conference, volume 2, SPLC. ACM, New York, pp 1–5. doi:10.1145/2019136.2019159

Passos L, Czarnecki K, Apel S, Wąsowski A, Kästner C, Guo J (2013) Feature-oriented software evolution. In: 7th international workshop on variability modelling of software-intensive systems, vamos. ACM, New York, pp 1–8. doi:10.1145/2430502.2430526

Peng X, Xing Z, Tan X, Yu Y, Zhao W (2013) Improving feature location using structural similarity and iterative graph mapping. J Syst Softw 86(3):664–676. doi:10.1016/j.jss.2012.10.270

Polzer A, Merschen D, Botterweck G, Pleuss A, Thomas J, Hedenetz B, Kowalewski S (2012) Managing complexity and variability of a model-based embedded software product line. Innov Syst Softw Eng 8(1):35–49. doi:10.1007/s11334-011-0174-z

Ramos MA, Penteado RA (2008) Embedded software revitalization through component mining and software product line techniques. J Universal Comput Sci 14(8):1207–1227

Romero D, Urli S, Quinton C, Blay-Fornarino M, Collet P, Duchien L, Mosser S (2013) SPLEMMA: A generic framework for controlled-evolution of software product lines. In: 5th international workshop on model-driven approaches in software product line engineering; 4th workshop on scalable modeling techniques for software product lines, MAPLE/SCALE. New York, pp 59–66. doi:10.1145/2499777.2500709

Rubin J (2014) Cloned product variants: From ad-hoc to well-managed software reuse. Ph.D. thesis, University of Toronto Graduate Department of Computer Science

Rubin J, Chechik M (2010) From products to product lines using model matching and refactoring. In: 2nd international workshop on model-driven approaches in software product line engineering (MAPLE), collocated with the 14th international software product line conference, pp 1–8

Rubin J, Chechik M (2012) Combining related products into product lines. In: 15th International Conference on Fundamental Approaches to Software Engineering, FASE, pp 285–300. Springer-Verlag, Berlin, Heidelberg. doi:10.1007/978-3-642-28872-2_20

Rubin J, Chechik M (2012) Locating distinguishing features using diff sets. In: 27th IEEE/ACM international conference on automated software engineering, ASE. ACM, New York, pp 242–245. doi:10.1145/2351676.2351712

Rubin J, Czarnecki K, Chechik M (2013) Managing cloned variants: a framework and experience. In: 17th international software product line conference, SPLC. ACM, New York, pp 101–110. doi:10.1145/2491627.2491644

Rubin J, Czarnecki K, Chechik M (2015) Cloned product variants: from ad-hoc to managed software product lines. Int J Softw Tools Technol Trans 17(5):627–646. doi:10.1007/s10009-014-0347-9

Rubin J, Kirshin A, Botterweck G, Chechik M (2012) Managing forked product variants. In: 16th international software product line conference - volume 1, SPLC. ACM, New York, pp 156–160 doi:10.1145/2362536.2362558

Ryssel U, Ploennigs J, Kabitzsch K (2011) Extraction of feature models from formal contexts. In: 15th international software product line conference - volume 2, SPLC. ACM, New York, pp 1–8. doi:10.1145/2019136.2019141

Sampath P (2013) An elementary theory of product-line variations. Formal Aspects of Computing pp 1–33. doi:10.1007/s00165-013-0276-5

Santos A, Gaia F, Figueiredo E, Neto PS, Araújo JA (2013) Test-based SPL extraction: An exploratory study. In: 28th symposium on applied computing, SAC. ACM, New York, pp 1031–1036. doi:10.1145/2480362.2480559

Schulze M, Mauersberger J, Beuche D (2013) Functional safety and variability: Can it be brought together? In: 17th international software product line conference, SPLC. ACM, New York, pp 236–243. doi:10.1145/2491627.2491654

Segura S, Parejo J, Hierons RM, Benavides D, Ruiz-Cortés A, De andalucía EDLJ (2012) ETHOM: An evolutionary algorithm for optimized feature models generation. Tech. Rep. ISA-2012-TR-01, Applied Software Engineering Research Group, Department of Computing Languages and tab9 University of Sevilla

Seidl C, Heidenreich F, Aßmann U (2012) Co-evolution of models and feature mapping in software product lines. In: 16th international software product line conference - volume 1, SPLC. ACM, New York, pp 76–85. doi:10.1145/2362536.2362550

Shao J, Wu W, Geng P (2013) An improved approach to the recovery of traceability links between requirement documents and source codes based on latent semantic indexing. In: Murgante B, Misra S, Carlini M, Torre C, Nguyen HQ, Taniar D, Apduhan B, Gervasi O (eds) Computational Science and Its Applications (ICCSA), Lecture Notes in Computer Science, vol. 7975, pp 547–557. Springer Berlin Heidelberg. doi:10.1007/978-3-642-39640-3_40

She S (2013) Feature model synthesis. Ph.D. thesis, University of Waterloo Electrical and Computer Engineering Department

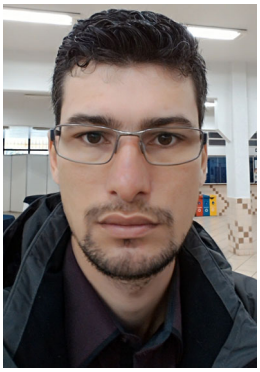
She S, Lotufo R, Berger T, Wąsowski A, Czarnecki K (2011) Reverse engineering feature models. 33rd international conference on software engineering, ICSE. ACM, New York, pp 461–470. doi:10.1145/1985793.1985856

She S, Ryssel U, Andersen N, Wąsowski A, Czarnecki K (2014) Efficient synthesis of feature models. Inform Softw Technol 0(0):1–22. doi:10.1016/j.infsof.2014.01.012

Stoermer C, O'Brien L (2001) MAP - Mining architectures for product line evaluations. In: Working IEEE/IFIP conference on software architecture (WICSA), pp 35–44. doi:10.1109/WICSA.2001.948405

Stuikys V, Valincius K (2011) A domain understanding through context-based feature modelling: a research framework. In: 17th international conference on information and software technologies (ICIST), pp 141–148

Tang Y, Leung H (2015) Top-down feature mining framework for software product line. In: 17th international conference on enterprise information systems, pp 71–81. doi:10.5220/0005370300710081
Trifu M (2010) Tool-supported identification of functional concerns in object-oriented code. Ph.D. thesis, Karlsruhe Institute of Technology
Valinčius K, Štuikys V, Damaševičius R (2013) Understanding of e-commerce is through feature models and their metrics to support re-modularization. International Journal on Computer Science and Information Systems (IADIS) 8(1):47–65
Van Der Storm T (2007) Generic feature-based software composition. In: 6th international conference on software composition (SC), SC'07. Springer-Verlag, Berlin, Heidelberg, pp 66–80
Weston N, Chitchyan R, Rashid A (2009) A framework for constructing semantically composable feature models from natural language requirements. In: 13th international software product line conference, SPLC. Carnegie Mellon University, Pittsburgh, pp 211–220
Xue Y (2012) Reengineering legacy software products into software product line. Ph.D. thesis, National University of Singapore Department of Computer Science
Xue Y, Xing Z, Jarzabek S (2010) Understanding feature evolution in a family of product variants. In: 17th working conference on reverse engineering (WCRE), pp 109–118
Xue Y, Xing Z, Jarzabek S (2012) Feature location in a collection of product variants. In: 19th working conference on reverse engineering (WCRE), pp 145–154. doi:10.1109/WCRE.2012.24
Yang Y, Peng X, Zhao W (2009) Domain feature model recovery from multiple applications using data access semantics and formal concept analysis. In: 16th working conference on reverse engineering (WCRE), pp 215–224. doi:10.1109/WCRE.2009.15
Yu Y, Wang H, Yin G, Liu B (2013) Mining and recommending software features across multiple web repositories. ACM, New York, doi:10.1145/2532443.2532453
Zhang G, Shen L, Peng X, Xing Z, Zhao W (2011) Incremental and iterative reengineering towards software product line: an industrial case study. In: 27th IEEE international conference on software maintenance (ICSM), pp 418–427. doi:10.1109/ICSM.2011.6080809
Ziadi T, Frias L, da Silva M, Ziane M (2012) Feature identification from the source code of product variants. In: 16th european conference on software maintenance and reengineering (CSMR), pp 417–422. doi:10.1109/CSMR.2012.52
Ziadi T, Henard C, Papadakis M, Ziane M, Le Traon Y (2014) Towards a language-independent approach for reverse-engineering of software product lines. In: 29th Symposium On Applied Computing (SAC) pp 1064–1071



Wesley K. G. Assunção received a bachelor's degree in Information Systems from Faculdade Sul Brasil in 2006 and the MSc degree in 2012 from Federal University of Paraná (UFPR), Brazil. He is currently PhD candidate at the Post-graduation Program in Informatics of Federal University of Paraná (UFPR), being a member of Research Group on Software Engineering - GRES. His areas of interest are: Software Testing, Software Product Lines, Search Based Software Engineering and Multi-Objective Evolutionary Algorithms.

Roberto E. Lopez-Herrejon is an Associate Professor at the Department of Software Engineering and Information Technology of the École de Technologie Supérieure of the University of Quebec in Montreal, Canada. Prior he was a senior postdoctoral researcher at the Johannes Kepler University in Linz, Austria. He was an Austrian Science Fund (FWF) Lise Meitner Fellow (2012–2014) at the same institution. From 2008 to 2014 he was an External Lecturer at the Software Engineering Masters Programme of the University of Oxford, England. From 2010 to 2012 he held an FP7 Intra-European Marie Curie Fellowship sponsored by the European Commission. He obtained his Ph.D. from the University of Texas at Austin in 2006, funded in part by a Fulbright Fellowship sponsored by the U.S. State Department. From 2005 to 2008, he was a Career Development Fellow at the Software Engineering Centre of the University of Oxford sponsored by Higher Education Founding Council of England (HEFCE). His main expertise is in software customization, software product lines, and search based software engineering.

Lukas Linsbauer is currently a PhD student at the Institute for Software Systems Engineering at the Johannes Kepler University (JKU) in Linz, Austria under the supervision of Prof. Alexander Egyed and Dr. Roberto Erick Lopez-Herrejon. He received his master's degree in computer science from the JKU after only four years of study for each of which he received a merit scholarship. His research interests are in traceability, software product lines, variability modeling and management, and highly variable and configurable systems.

Silvia R. Vergilio received the MS (1991) and DS (1997) degrees from University of Campinas, UNICAMP, Brazil. She is currently at the Computer Science Department at the Federal University of Paraná, Brazil, where she has been a faculty member since 1993. She has been involved in several projects and her research interests are in the areas of Software Engineering, such as: software testing, software architecture, software metrics, and search-based software engineering.

Alexander Egyed heads the Institute for Software Systems Engineering (ISSE) at the Johannes Kepler University, Austria. He received his Doctorate from the University of Southern California, USA and previously worked many years in industry before joining academia. Dr. Egyed was recognized among the Top 10 scholars in software engineering and his work has received numerous awards.

Appendix B

Multi-objective reverse engineering of variability safe feature models based on code dependencies of system variants

Empir Software Eng
DOI 10.1007/s10664-016-9462-4

Multi-objective reverse engineering of variability-safe feature models based on code dependencies of system variants

Wesley K. G. Assunção[1,2] · Roberto E. Lopez-Herrejon[3] · Lukas Linsbauer[4] · Silvia R. Vergilio[1] · Alexander Egyed[4]

Abstract Maintenance of many variants of a software system, developed to supply a wide range of customer-specific demands, is a complex endeavour. The consolidation of such variants into a Software Product Line is a way to effectively cope with this problem. A crucial step for this consolidation is to reverse engineer feature models that represent the desired combinations of features of all the available variants. Many approaches have been proposed for this reverse engineering task but they present two shortcomings. First, they use a single-objective perspective that does not allow software engineers to consider design trade-offs. Second, they do not exploit knowledge from implementation artifacts. To address

Communicated by: Mark Harman

✉ Wesley K. G. Assunção
wesleyk@inf.ufpr.br

Roberto E. Lopez-Herrejon
roberto.lopez@etsmtl.ca

Lukas Linsbauer
lukas.linsbauer@jku.at

Silvia R. Vergilio
silvia@inf.ufpr.br

Alexander Egyed
alexander.egyed@jku.at

1 DINF, Federal University of Paraná (UFPR), CP: 19081, CEP: 81.531-980, Curitiba, Brazil

2 COTSI, Federal University of Technology - Paraná (UTFPR), Cristo Rei Street, 19. CEP: 85.902-490, Toledo, Brazil

3 Department of Software Engineering and IT, École de Technologie Supérieure, (ÉTS), Notre-Dame Street Ouest. 1100, H3C 1K3 Montreal, Canada

4 ISSE, Johannes Kepler University Linz (JKU), Altenbergerstr. 69, 4040 Linz, Austria

Published online: 20 October 2016

these limitations, our work takes a multi-objective perspective and uses knowledge from source code dependencies to obtain feature models that not only represent the desired feature combinations but that also check that those combinations are indeed well-formed, i.e. variability safe. We performed an evaluation of our approach with twelve case studies using NSGA-II and SPEA2, and a single-objective algorithm. Our results indicate that the performance of the multi-objective algorithms is similar in most cases and that both clearly outperform the single-objective algorithm. Our work also unveils several avenues for further research.

Keywords Reverse engineering · Feature models · Multi-objective evolutionary algorithms · Empirical evaluation

1 Introduction

Software Product Lines (SPLs) are families of software products with focus on reuse of artefacts (Batory et al. 2004). SPLs have been receiving wide attention in the industry due to their advantages, among them, higher software quality and shorter time-to-market for new products (van d. Linden et al. 2007). However, there are some challenges on the adoption of SPLs in industrial settings. One of the main challenges is how to consolidate existing system variants into a SPL. To tackle such a challenge, reverse engineering strategies have been proposed to deal with different stages of the SPL development (Assunção and Vergilio 2014). The starting point to this reverse engineering process is the extraction of *Feature Models (FMs)*, the de facto standard for modeling variability and commonality in SPLs (Benavides et al. 2010), which denote the set of valid configurations of features that constitute the products of a SPL.

In recent years several approaches to reverse engineer FMs have been proposed. They are based on configuration scripts (She et al. 2011), propositional logic expressions (Czarnecki and Wasowski 2007; She et al. 2014), natural language (Weston et al. 2009), ad hoc algorithms (Acher et al. 2012; Haslinger et al. 2011, 2013), and search-based techniques (Linsbauer et al. 2014; Lopez-Herrejon et al. 2012, 2015; Thianniwet and Cohen 2015). However, they present two main limitations. They do not take a multi-objective perspective to capture the trade-offs that software engineers must make for the reverse engineering task, and do not exploit any knowledge on how the system variants are actually implemented.

A multi-objective perspective has advantage in situations where there are conflicts among the goals of the software engineer. For instance, obtaining an FM that represents a set of desired product configurations can lead to a model that also generates a surplus of configurations, which are not desired. Nevertheless, if we tweak the FM to avoid such additional configurations we might loose some desired configurations. Here a multi-objective algorithm aims to optimizing both goals and enables the engineer to make a decision based on an analysis of the different trade-offs of importance in the problem domain. In addition, the use of knowledge available in implementation artefacts is an important characteristic because we can generate FMs in accordance with the already existing software variants.

To cope with these limitations, our previous work presented an approach to reverse engineer FMs based on the multi-objective algorithm NSGA-II (Assunção et al. 2015). This work used a graph to represent dependencies in the source code artifacts of the existing system variants, and provided software engineers with sets of FMs with different trade-offs. In this paper we extend our previous work by including: *i)* a second multi-objective evolutionary algorithm, namely *Strength Pareto Evolutionary Algorithm (SPEA2)* (Coello et al.

2007), *ii)* a single-objective evolutionary algorithm that relies on a genetic programming representation for baseline comparison, *iii)* a detailed description of our problem representation and evolutionary operators, *iv)* seven new case studies, and *v)* a more thorough empirical evaluation and analysis. In summary, our current work addresses the following research questions:

- *RQ1: What are the benefits and advantages of using a multi-objective approach over a single-objective one to reverse engineer FMs?*
- *RQ2: How does the performance of NSGA-II and SPEA2 compare for reverse engineering FMs?*
- *RQ3: How could a multi-objective perspective be used in practice to support software engineers in the decision making process?*

The remainder of this paper is structured as follows: Section 2 presents the fundamental concepts of feature models, a running example, and the definitions regarding the dependency graphs. The details of the proposed approach are described in Section 3. The setup used in the evaluation of the proposed approach is found in Section 4. Section 5 has the results obtained in the evaluation and the corresponding analysis in order to answer the research questions. Related work is found in Section 6. The research avenues for future work and conclusions are presented respectively in Sections 7 and 8.

2 Background

In this section we present an overview of feature models, a running example to illustrate the details of our approach, and define more precisely the notions of dependencies and dependency graphs which we use to represent the source code dependencies. We rely on these definitions for describing our multi-objective approach in Section 3.

2.1 Feature Models

Feature Models (FMs) are widely used to model the different combinations of features in a SPL (Kang et al. 1990), and are key for supporting variability and commonality management (Benavides et al. 2010). These models follow a tree-like structure where features are depicted as boxes, labelled with the feature name, which are connected by lines with their children features.

An FM always has a *root* feature that is present in all products of the SPL. The other features can be of type *mandatory* or *optional*, or a set of features can be organized in groups as *alternative groups* or *or groups*. A mandatory feature is always selected when its parent feature is selected. An optional feature may or may not be selected when its parent is selected. The mandatory feature is represented with filled circle at the end of the relation line and the optional feature is represented with an empty circle. These two types of features are respectively presented in Fig. 1a and b. An *alternative group* indicates that when the parent feature of the group is selected then *exactly* one feature of the group must be selected. When the parent feature of an *or group* is selected then *at least* one feature of the group must be selected. The *alternative* group is represented by an empty arc and the *or* group is represented by a filled arc, as illustrated in Fig. 1c and d, respectively.

In addition to the hierarchical relation between the features, the valid configuration of features can also be restricted by some additional constraints, called *Cross-Tree Constraints*

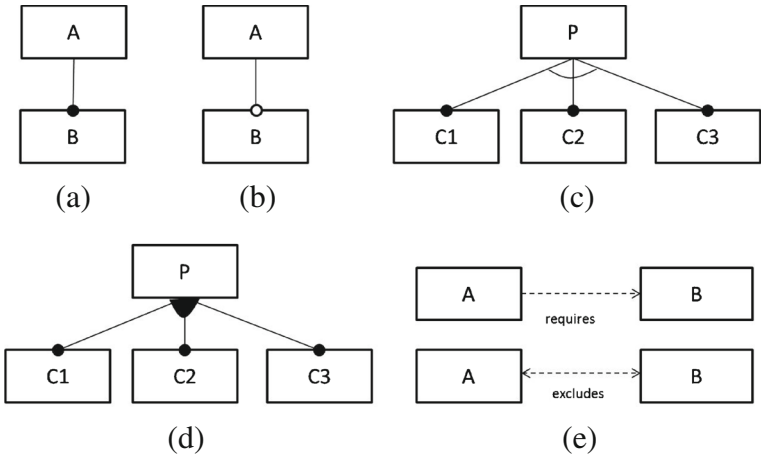


Fig. 1 Feature models graphical notation

(CTCs) (Benavides et al. 2010). The most common types of CTCs are *requires* and *excludes*. If a feature `A` is selected and requires feature `B`, then feature `B` must also be selected. If a feature `A` excludes feature `B` then these two features cannot both be selected in the same feature combination.

These two types of CTCs are usually respectively represented by single and double-arrow dashed lines as shown in Fig. 1e.

2.2 Running Example

In this subsection we present a running example to illustrate the details of our approach. We selected a set of variants of a drawing application. Our goal is to use these variants as a starting point to obtain a product line called *Draw Product Line (DPL)*. This application offers users the ability to handle a drawing area (feature `BASE`), draw lines (feature `LINE`), draw rectangles (feature `RECT`), draw filled rectangles (feature `FILL`), select a color for the line or rectangle (feature `COLOR`), and clean the drawing area (feature `WIPE`). With these six features we have a total number of 16 variants of the drawing application, presented in Table 1. The symbol ✓ indicates the selected features. We refer to each feature combination as a *feature set*, formally defined as Linsbauer et al. (2014):

Definition 1 Feature Set A *feature set* is a 2-tuple $[\texttt{sel},\overline{\texttt{sel}}]$ where `sel` and $\overline{\texttt{sel}}$ are respectively the set of selected and not-selected features of a system variant. Let `FL` be the list of features of a feature model, such that $\texttt{sel}, \overline{\texttt{sel}} \subseteq \texttt{FL}$, $\texttt{sel} \cap \overline{\texttt{sel}} = \emptyset$, and $\texttt{sel} \cup \overline{\texttt{sel}} = \texttt{FL}$.

2.3 Source Code Dependency Graphs

One of the main contributions of our approach is to use knowledge from implementation artefacts, besides the feature sets, to reverse engineer FMs. Based on this knowledge, we evaluate *variability safety* of FMs, a property that guarantees that the feature sets denoted by

Table 1 Feature sets for DPL

Products	BASE	LINE	RECT	COLOR	FILL	WIPE
$Product_1$	✓	✓				✓
$Product_2$	✓	✓		✓		
$Product_3$	✓	✓	✓	✓		
$Product_4$	✓	✓	✓	✓	✓	
$Product_5$	✓	✓	✓			✓
$Product_6$	✓	✓				
$Product_7$	✓	✓		✓		✓
$Product_8$	✓	✓	✓			
$Product_9$	✓		✓			
$Product_{10}$	✓		✓			✓
$Product_{11}$	✓		✓	✓		
$Product_{12}$	✓		✓	✓	✓	
$Product_{13}$	✓		✓	✓		✓
$Product_{14}$	✓		✓	✓	✓	✓
$Product_{15}$	✓	✓	✓	✓		✓
$Product_{16}$	✓	✓	✓	✓	✓	✓

an FM are structurally well-formed according to the source code. In particular, we exploit the information of existing dependencies between source code fragments, as described next.

To represent the dependencies existing in the source code of the different system variants we use a weighted directed graph. To create this graph we use the terminology and the tool from our previous work (Fischer et al. 2014; Linsbauer et al. 2013). We identify *modules* of two kinds:

Definition 2 Base Module A *base* module implements a feature regardless of the presence or absence of any other features and is denoted with the feature name written in lowercase.

Definition 3 Derivative Module A *derivative* module $m = \delta^n(c_0, c_1, ..., c_n)$ implements feature interactions, where c_i is F (if feature F is selected) or $\neg$F (if not selected), and n is the order of the derivative.

These two types of modules are the basis of our extraction algorithm (Linsbauer et al. 2013). This algorithm computes traces from the modules to their implementing source code fragments and identifies dependencies between the modules that have dependencies in their implementations. The algorithm considers any granularity of the implementation artefacts, from class level to statement level. Figure 2 presents some examples of traces computed by the algorithm. The traces are indicated using comments at the end of the lines. For example, the field defined in Line 5 traces to the base module $Color$ of the corresponding feature Color. This source code fragment is present in all variants with feature Color, regardless the existence of other features. Another example of a base module is observed in Line 7 of the example. This method header and most of its implementation will be included in class Canvas whenever feature Line is present, regardless of any other feature. However, in Line 10 of the method we can observe a derivative module $\delta^1(Color, Line)$, which means that the corresponding line of code will be part of its containing method only when the variant has both features Color and Line.

```
1 public class Canvas ... {
2   List<Shape> shapes = new LinkedList<Shape>();
3   Point start;
4   Line newLine = null; // Line
5   Color color = Color.BLACK; // Color
6   ...
7   void mousePressedLine(MouseEvent e) { // Line
8    if (newLine == null) {
9       start = new Point(e.getX(), e.getY());
10      newLine=new Line(color,start);//δ1(Color, Line)
11      shapes.add(newLine);
12   }
13  }
14 }
```

Fig. 2 Source code snippet for DPL example

In order to more formally define dependencies and their representation as graphs, we first introduce the concept of module expression as follows:

Definition 4 Module Expression A module expression is the propositional logic representation of modules. For a base module b, the module expression is its own literal b. For a derivative module $m = \delta^{n}(c_0, c_1, ..., c_n)$ its module expression corresponds to $c_0 \wedge c_1 \wedge ... \wedge c_n$.

As an illustration, the module expression of base module `Color` is $Color$. For the derivative module $\delta^{1}(\texttt{Color}, \texttt{Line})$, which indicates the interaction between features `Color` and `Line`, the module expression representation is $Color \wedge Line$.

The traces produced by our trace extraction algorithm are used to identify the dependencies between fragments of source code. These dependencies and the dependency graph they form are formally defined next.

Definition 5 Dependency A dependency establishes a requirement relationship between two sets of modules and it is denoted with a three-tuple `(from, to, weight)`, where `from` and `to` each are a set of modules (or module expressions) of the related modules, and `weight` expresses the strength of the dependency, i.e. the number of dependencies of structural elements in modules `from` on structural elements in modules `to`.

We use the dot (`.`) operator to refer to elements of a tuple, e.g. the weight of a dependency `dep` is denoted by `dep.weight`. A dependency's propositional logic representation is defined as:

$$\bigvee_{mfrom \in dep.from} mfrom \Rightarrow \bigwedge_{mto \in dep.to} mto$$

Definition 6 Dependency Graph A dependency graph is a set of dependencies, where each node in the graph corresponds to a set of modules (or module expressions), and every edge in the graph corresponds to a dependency as defined above. Edges are annotated with natural numbers that represent the dependencies' weights.

Now we recall our running example of the drawing application, Fig. 3 presents the dependency graph considering all its feature sets. To avoid clutter only the lowest order modules are presented, since they are the most relevant to our approach. Self-dependencies are removed from the graph for better readability. To make clear the different kinds of modules,

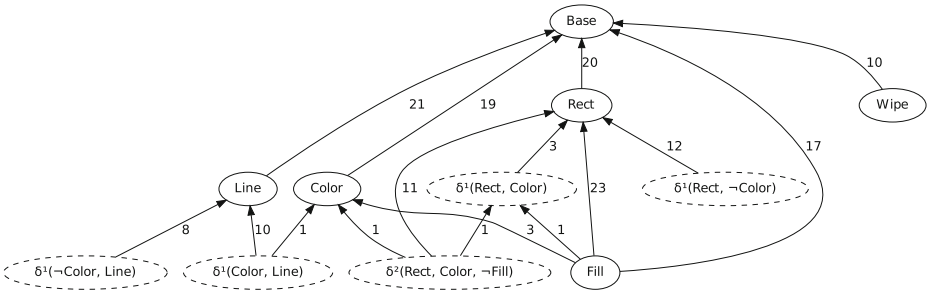


Fig. 3 Dependency graph for DPL

the base modules have solid borders and the derivative modules have dashed borders. In the figure we can observe that the strongest dependencies (i.e. those with the highest weights) are those that go to base modules, e.g. `Fill` to `Rect`, and the core feature `Base` has the largest number of incoming dependencies. As mentioned before, weights in the graph represent the number of structural dependencies between source code elements belonging to different features. For instance, in Fig. 3 there are 10 dependencies from source code elements of `WIPE` to source code elements of `BASE`. These dependencies are field accesses (4) and containment relationships (6), i.e a field belongs to a class.

Alternatively the dependency graph can be represented as a dependency matrix, as presented in Table 2. The rows are the dependencies, the first column is a dependency identification for easy reference, the second and third columns have the modules of the dependency in the order *from* to *to*, respectively. The weight of the dependency is displayed in the fourth column. In the fifth column the weight is normalized to keep the sum of all

Table 2 Dependency matrix for DPL

ID	From	To	Weight	Normalized
1	Line	Base	21	0.1304
2	Wipe	Base	10	0.0621
3	Color	Base	19	0.1180
4	Rect	Base	20	0.1242
5	Fill	Base	17	0.1056
6	Fill	Rect	23	0.1429
7	Fill	Color	3	0.0186
8	Fill	δ^1(Rect, Color)	1	0.0062
9	δ^1(Rect, ¬Color)	Rect	12	0.0745
10	δ^1(Rect, Color)	Rect	3	0.0186
11	δ^1(¬Color, Line)	Line	8	0.0497
12	δ^1(Color, Line)	Color	1	0.0062
13	δ^1(Color, Line)	Line	10	0.0621
14	δ^2(Rect, Color, ¬Fill)	δ^1(Rect, Color)	1	0.0062
15	δ^2(Rect, Color, ¬Fill)	Rect	11	0.0683
16	δ^2(Rect, Color, ¬Fill)	Color	1	0.0062
		Total:	161	1.0000

weights of the graph equal to 1.0. This normalization enables a better interpretation of the values in the optimization process.

To illustrate the propositional logic representation of dependencies let us use again the source code shown in Fig. 2. Firstly consider the dependency that exists between the module $\delta^1(Color, Line)$ and the module $Color$. This dependency exists because the field `color` defined in Line 5 belongs to the module $Color$ and it is used by `newLine = new Line(color,start);` in Line 10 which belongs to the module $\delta^1(Color, Line)$. The propositional logic expression for this dependency is $(Color \wedge Line) \Rightarrow Color$. In the same code snippet of Fig. 2 we can see that module $\delta^1(Color, Line)$ depends on module $Line$, because the statement in Line 10 is contained in the method `void mousePressedLine(MouseEvent e)` which belongs to module $Line$. In this case, the propositional logic expression is $(Color \wedge Line) \Rightarrow Line$.

An important point is to observe that the propositional logic constraints of some dependencies are tautologies, hence they always hold. The two above examples illustrate this situation, since $(Color \wedge Line \Rightarrow Line) \Leftrightarrow TRUE$. This indicates that the implementation artefacts are consistent with the feature combinations represented in the FM.

3 Multi-Objective Approach to Reverse Engineer Feature Models

In this section we describe the details of our multi-objective search-based approach which relies on and extends upon our previous work (Linsbauer et al. 2014). We further describe the requirements identified by Harman et al. (2012) to implement a search-based solution: (i) an adequate representation of solutions, (ii) a set of operators to improve the solutions and explore the search space, and (iii) an adequate way to evaluate the quality of the solutions, the fitness functions.

3.1 Feature Model Representation

The feature model representation uses a simplified version of the SPLX metamodel.[1] This metamodel, presented in Fig. 4, defines both structure and semantic of the FMs. In the figure, the elements in the left part describe the tree-like structure of the FM and the elements in the right describe the CTCs between the features. The tree nodes *Root*, *Mandatory*, *Optional*, and *GroupedFeature* inherit from *Feature*. The tree is composed by exactly one *Root* feature. Features *Mandatory* and *Optional* have the cardinality of zero or more. The tree can also have an arbitrary number of *Alternative* and *Or* groups and each group must have at least one *GroupedFeature*. In the right part of the figure we can observe that an FM has exactly one *ConstraintSet*. This *ConstraintSet* describes the propositional formula in CNF. Zero or more *Constraint* are acceptable, each constraints is a clause in a CNF expression. Each *Constraint* has exactly one *OrClause* that has at least one *Literal*. A literal can either be an *Atom* which refers directly to a feature, or a *Not* which refers to an *Atom*.

Following this representation, the initial population is created by generating random feature trees and random CTCs. Some additional domain constraints are also taken into account, they are presented in the next subsection. The tools FaMa (Benavides et al. 2007) and BeTTy (Segura et al. 2012) were used to create the initial population.

[1] http://www.splot-research.org/.

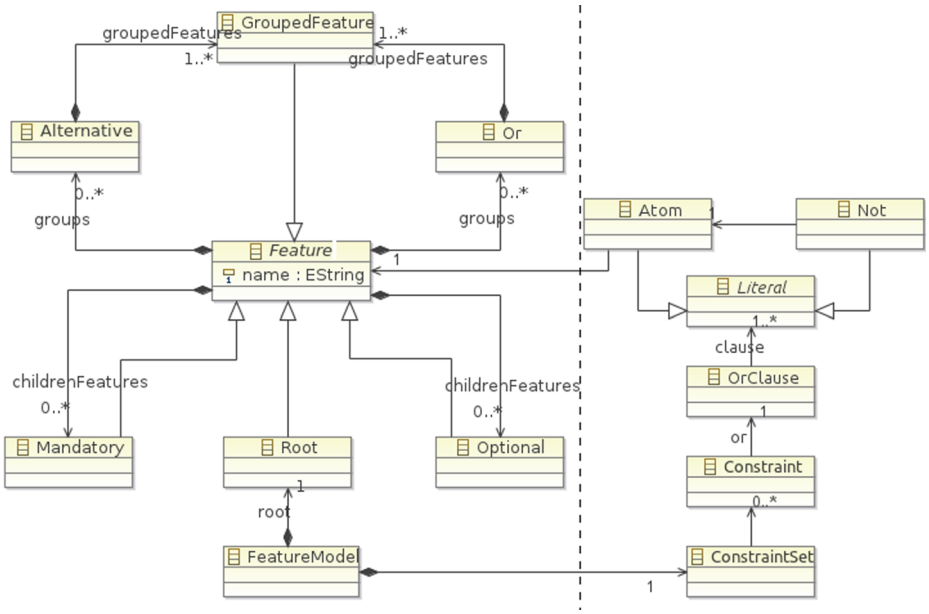


Fig. 4 Feature model metamodel, extracted from Linsbauer et al. (2014)

3.2 Evolutionary Operators

We adopted the evolutionary operators from our previous work (Linsbauer et al. 2014), and employ standard tournament selection as selection operator. There are some domain constraints that should be taken into account in the evolutionary process to guarantee the semantics of FMs and to avoid generating invalid solutions:

- Each feature is identified by its name, so every feature appears exactly once in the FM tree;
- All FMs have a fixed set of feature names, so in different FMs only the relations between features are different;
- CTCs can only be either *requires* or *excludes*, i.e. exactly two literals per clause with at least one being negated;
- CTCs must not contradict each other, i.e. the corresponding CNF of the entire constraint set must be satisfiable;
- There is a maximum number of CTCs (given as a percentage of the number of features) that must not be exceeded.

Our domain constraints do not consider the rare case of contradictions between CTCs and the FM tree for which the detection and repair is computationally expensive. For individuals in that case, we let the evolutionary process itself weed them out because of their bad fitness value.

3.2.1 Mutation

The mutation operator applies small changes in randomly selected parts of the tree or in the CTCs of the feature model. The mutation probability is used to decide if the change is

applied in the tree part, in the CTCs, or in both. The kind of change is randomly selected from the following lists:

- Mutations performed on the tree:
  - Randomly swaps two features in the feature tree;
  - Randomly changes an *Alternative* relation to an *Or* relation or vice-versa;
  - Randomly changes an *Optional* or *Mandatory* relation to any other kind of relation (*Mandatory*, *Optional*, *Alternative*, *Or*);
  - Randomly selects a subtree in the feature tree and puts it somewhere else in the tree without violating the metamodel or any of the domain constraints.
- Mutations performed on the CTCs:
  - Adds a new, randomly created CTC that does not contradict the other CTCs and does not already exist;
  - Randomly removes a CTC.

3.2.2 Crossover

Just like the mutation operator, the crossover must generate offspring in conformance to the metamodel and to the domain constraints. The steps of the crossover process are:

1. The offspring is initialized with the root feature of $Parent_1$. If the root feature of $Parent_2$ is a different one then it is added to the offspring as a mandatory child feature of its root feature.
2. Traverse the first parent depth first starting at the *root* node and add to the offspring a random number r of features that are not already contained by appending them to their respective parent feature already contained in the offspring using the same relation type between them (the parent feature of every visited feature during the traversal is guaranteed to be contained in the offspring due to the depth first traversal order).
3. Traverse the second parent exactly the same way as the first one.
4. Go to Step 2 until every feature is contained in the offspring.

The second child is obtained by performing the same process but with reverse parents, i.e. the position of the parents is swapped.

The CTCs offspring are obtained by merging all the constraints of both parents and then randomly selecting a subset of CTCs that are assigned to the first offspring and the remaining to the second offspring.

3.3 Multi-Objective Perspective

In this section we describe the three objective functions used in our approach and present an illustrative example.

3.3.1 Auxiliary Functions Definitions

In order to compute the objective functions of our approach we need some auxiliary functions. Let us consider $\mathcal{FM}$ as the universe of feature models, $\mathcal{SFS}$ the universe of set of feature sets, and sfs a set of feature sets defined by the software engineer. An example

of sfs is the feature sets presented in Table 1 for the drawing application. Based on this terminology, we introduce the function *featureSets*:

Definition 7 featureSets. Function *featureSets* returns the set of feature sets denoted by a feature model.

$$featureSets : \mathcal{FM} \rightarrow \mathcal{SFS}$$

To measure the variability safety of an FM we have to check if its feature sets are in conformance with the dependencies of the dependency graph. To check this we define the function *holds*:

Definition 8 holds Function $holds(dep, fs)$ returns 1 if dependency dep holds on the feature set (of a system variant) fs and 0 otherwise. A dependency dep holds for a feature set fs if:

$$\left(\bigwedge_{f \in fs.sel} f \wedge \bigwedge_{g \in fs.\overline{sel}} \neg g \right) \Rightarrow (dep.from \Rightarrow dep.to)$$

To illustrate this function recall our running example. Let us consider the dependency $Fill \Rightarrow Rect$ and a system variant with the features `Base` and `Line`. In this case the function *holds* returns 1, since the propositional logic formula $(Base \wedge Line \wedge \neg Fill \wedge \neg Rect \wedge \neg Wipe \wedge \neg Color) \Rightarrow (Fill \Rightarrow Rect)$ is true.

3.3.2 Fitness Functions Definitions

Considering the two auxiliary functions presented before we are able to introduce the three objective functions of our approach. The first two measures are based on information retrieval metrics, for further details refer to Manning et al. (2008), and the third measure is based on variability safety.

Definition 9 Precision (P). Precision expresses how many of the feature sets denoted by a reverse engineered feature model fm are among the desired feature sets sfs.

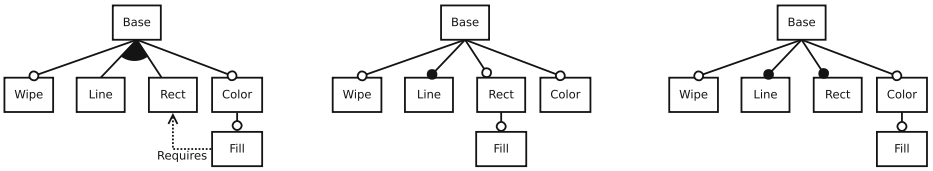
$$precision(sfs, fm) = \frac{|sfs \cap featureSets(fm)|}{|featureSets(fm)|}$$

Definition 10 Recall (R). Recall expresses how many of the desired feature sets are denoted by the reverse engineered feature model fm.

$$precall(sfs, fm) = \frac{|sfs \cap featureSets(fm)|}{|sfs|}$$

Definition 11 Variability Safety (VS). Variability Safety expresses the degree of *variability-safety* of a reverse engineered feature model fm with respect to a dependency graph dg.

$$variabilitySafety(fm, dg) = \sum_{dep \in dg} dep.weight \times \left(\frac{\sum_{fs \in featureSets(fm)} holds(dep, fs)}{|featureSets(fm)|} \right)$$



(a) FM1 (b) FM2 (c) FM3

P	$= 16/16 = 1.000$	P	$= 10/12 = 0.833$	P	$= 6/6 = 1.000$
R	$= 16/16 = 1.000$	R	$= 10/16 = 0.625$	R	$= 6/16 = 0.375$
VS	$= 1.000$	VS	$= 0.996$	VS	$= 1.000$

Fig. 5 Examples of extracted feature models for DPL

3.3.3 Fitness Functions Illustration

To illustrate our three objective functions let us consider the feature sets of sfs in Table 1, the dependency graph dg in Fig. 3, and the normalized weight values for dg from Table 2. Figure 5 presents three examples of FMs extracted from the drawing application variants. We computed the values of precision, recall and variability safety for these FMs.

In Fig. 5a the feature model FM1 is an ideal solution for the sfs in Table 1. This feature model has $|featureSets(FM1)| = |sfs| = 16$, leading to precision and recall equal to 1.000, which means that its valid configurations are exactly the same as the desired feature sets. The value of variability safety for this FM is also 1.000, indicating that the valid configurations of FM1 do not break any dependency.

The feature model FM2, presented in Fig. 5b, denotes $|featureSets(FM2)| = 12$ feature sets of which $|sfs \cap featureSets(FM2)| = 10$ are in the desired feature sets. With these values we have precision = 0.833 and recall = 0.625. Some feature sets denoted by this feature model do not satisfy all dependencies, leading to a value of variability safety = 0.996. To illustrate some broken dependencies we use the feature set $[\{Base, Line, Rect, Fill\}, \{Wipe, Color\}]$. For this feature set the dependency ID 7 (`Fill` $\Rightarrow$ `Color`), shown in Table 2, is not satisfied because the dependency indicates that when `Fill` is in the feature set the feature `Color` must also be included. When a dependency is not satisfied, its normalized weight is not added to the accumulated weight of the satisfied dependencies effectively decreasing the value of variability safety.

Let us now consider the feature model FM3 presented in Fig. 5c. This feature model denotes six feature sets and all of them are desired feature sets. Hence, $|featureSets(FM3)| = 6$ and $|sfs \cap featureSets(FM3)| = 6$, leading to precision = 1.000 and recall = 0.375. Furthermore, no dependency is broken by its feature sets, so the value of variability safety is 1.000. For instance, considering a single feature set $[\{Base, Line, Rect, Color, Fill\}, \{Wipe\}]$, the dependencies with IDs {1, 3, 4, 5, 6, 7, 8, 10, 12, 13} are satisfied because both the *from* modules and the *to* modules are contained. In addition, dependencies with IDs {2, 9, 11, 14, 15, 16} are also satisfied because the *from* modules are not part of the feature set. Once the *from* module is not included in the feature set, it is not required the *to* module be contained.

These illustrative examples show the possible diversity of values among the three objectives. From the decision maker point of view FM1 is the best one, since it has the best values for the three objectives. However, in most of the situations, ideal solutions like this do not necessarily exist, and hence many trade-offs must be considered.

Table 3 Algorithm's parameters

Parameter	GP	NSGA-II	SPEA2
Number of Generations	1000	1000	1000
Population Size	200	200	200
Archive Size	–	–	10
Crossover	0.7	0.7	0.7
Feature Tree Mutation	0.5	0.5	0.5
CTCs Mutation	0.5	0.5	0.5
Number of Elites	25 %	25 %	25 %
Selection Method	Tournament	Tournament	Tournament
Tournament Size	6	6	6
Maximium CTC Percentage for Builder[a]	0.1	0.1	0.1
Maximium CTC Percentage for Mutator[a]	0.5	0.5	0.5
Independent runs	30	30	30

[a]relative to number of features

4 Experimental Description

In this section we present our experimental setup and describe the case studies used for our evaluation. The implementation and data are available online for replication.[2]

4.1 Experimental Setup

In order to answer our research questions we perform an experiment, which is described as follows. As mentioned before, our focus is to solve the problem of reverse engineering of FMs using three objective functions: *Precision (P)*, *Recall (R)*, and *Variability Safety (VS)*. We designed these measures to be normalized values in the interval between 0 and 1, where the goal is to maximize the values of all objective functions. Hence, the ideal solution is P = 1.0, R = 1.0, and VS = 1.0.

In Section 3.1 we described the genetic programming representation we used. In addition to our previous work (Assunção et al. 2015), where we applied only the *Non-Dominated Sorting Genetic Algorithm (NSGA-II)* (Deb et al. 2002), here we consider two additional algorithms. The single-objective *Genetic Programming (GP)* algorithm from related work (Linsbauer et al. 2014), and the *Strength Pareto Evolutionary Algorithm (SPEA2)* (Zitzler et al. 2001), which is characterized by its external archive used to create the fronts of non-dominated solution in each generation. The GP algorithm was applied to serve as a baseline comparison. Our GP algorithm uses a weighted measure of precision and recall with same weight for both values, called F_1 based on information retrieval theory (Manning et al. 2008). The implementation is the same as in our previous work, for further details, refer to Linsbauer et al. (2014). SPEA2 was added because it is widely used with NSGA-II (Coello et al. 2007) and commonly applied in search-based software engineering approaches (Harman et al. 2012). For the experimentation we used ECJ Framework.[3] The parameter

[2]http://www.inf.ufpr.br/gres/IS/MORevEngFMs.zip.

[3]http://cs.gmu.edu/~eclab/projects/ecj/.

Table 4 Case studies overview

System	#F	#P	LoC	#Nodes	#Edges
ArgoUML	11	256	264K–344K	49	114
DPL	6	16	282–473	12	27
GOL	15	65	874–1.9K	12	24
VOD	11	32	4.7K–5.2K	7	11
ZipMe	7	32	5K–6.2K	29	60
MM-V1	5	3	2.1K	5	4
MM-V2	6	6	2.3K–2.4K	6	6
MM-V3	7	12	2.3K–2.5K	9	12
MM-V4	8	24	2.6K–2.9K	10	14
MM-V5	9	48	2.7K–3.8K	14	25
MM-V6	10	96	2.8K–4.1K	22	43
MM-V7	13	240	2.9K–4.3K	31	70

#F: Number of Features, #P: Number of Products, LoC: Lines of Code, #Nodes: Number of Nodes in the Dependency Graph, #Edges: Number of Edges, i.e. Dependencies, in the Dependency Graph

settings used to configure the algorithms are shown in Table 3. We performed 30 independent runs for each algorithm for each case study. The runs were performed on a machine with an Intel® Core^{TM} i7-4900MQ CPU with 2.80 GHz, 16 GB of memory, and running on a Linux platform.

4.2 Case Studies

To evaluate the proposed approach we used the case studies presented in Table 4. ArgoUML is an open source tool for UML modelling (Couto et al. 2011). Draw Product Line (DPL), briefly presented in our running example, is a small drawing application. Game Of Life (GOF) is a customizable game. Video On Demand (VOD) implements video-on-demand

Table 5 Average runtime per run

System	GP			NSGA-II			SPEA2		
	min	sec	msec	min	sec	msec	min	sec	msec
ArgoUML		25	275	13	59	911	17	35	300
DPL		23	512		35	802	1	30	583
GOL		56	934	2	15	991	3	3	531
VOD			906		1	882		2	247
ZipME		1	131	1	50	576	2	57	495
MM-V1		4	816		10	468		20	258
MM-V2		8	502		15	321		38	451
MM-V3		17	212		22	325		45	628
MM-V4		29	327		30	624		58	208
MM-V5		52	962		59	342	1	44	310
MM-V6	1	27	442	3	28	598	4	18	747
MM-V7	2	35	35	10	53	762	11	2	444

min = minutes, sec = seconds, msec = milliseconds

streaming. ZipMe is an application for files compression. MobileMedia (MM) is an application to manipulate media files, such as photo, music, and video, on mobile devices. In our evaluation we used seven versions of MobileMedia (Figueiredo et al. 2008).

5 Results and Analysis

The average runtime per run of each algorithm is presented in Table 5. GP is the fastest algorithm in all case studies, respectively followed by NSGA-II and SPEA2. As expected, the multi-objective algorithms performed slower than the single-objective GP because of the computation to compose the Pareto fronts in each generation. Nonetheless, next we elaborate on the advantages of a multi-objective approach for our reverse engineering task.

For the analysis of results obtained by the algorithms we computed two different sets of solutions. Our terminology is based on our previous work (see Assunção et al. 2014) and standard multi-objective optimization literature (Coello et al. 2007).

- *Pareto Front True* (PF_{True}): since we do not know the real Pareto Front True, we use PF_{True} as an approximation of the best solutions. This set consists of the best solutions reached by all algorithms for each case study. These best solutions are found by merging all the solutions of all runs of the three algorithms together, then leaving only the non-dominated solutions. The cardinality of PF_{True} for each case study is presented in the second column of Table 6.
- *Pareto Front Known* (PF_{Known}): this set contains the best solutions reached by each algorithm for each case study. To compute PF_{Known} we merged all the solutions of all runs for each algorithm and then we keep only the non-dominated solutions. The cardinality of PF_{Known} for each case study and each algorithm is presented in the fourth column of Table 6.

5.1 Answering RQ1

Recall that RQ1's purpose is to find what the benefits are of a multi-objective perspective for reverse engineering FMs. We do so by comparing solutions obtained from multi-objective algorithms against solutions from a single-objective algorithm. From Table 6 we can observe that in seven case studies there is only one single solution in PF_{True}, so all the three objectives could be optimized independently. For the other five case studies with more than one solution there are conflicts among the objectives. Taking into account the seven versions of Mobile Media, we can observe that the three objectives became conflicting in MM-V6. Besides, we can note that MM-V6 and MM-V7 have a large number of solutions in comparison with the other case studies with cardinality larger than one in PF_{True}. We found out that this happens because MM-V6 introduced big changes in the source-code of MobileMedia (Figueiredo et al. 2008). As explained in Figueiredo et al. (2008), MM-V6 and MM-V7 introduced the two alternative features `Music` and `Video`, and the mandatory `Photo` feature was made optional leading to a big impact on the whole system. These changes lead to a larger number of nodes and edges, see Table 4, which makes the dependency graph more complex and impacts the number of solutions found.

For the analysis on the ability of each algorithm to find non-dominated solutions we consider two values presented in Table 6: (i) the number of solutions found by each algorithm, i.e. PF_{known}, that are in PF_{True}, fifth column; and (ii) the average number of solutions found per run that are/were in PF_{True}, sixth column. To illustrate the meaning of these

Table 6 Pareto fronts

System	PF_{True} cardinality	Search Algorithm	PF_{known} cardinality	# of solutions in PF_{True}	Average in PF_{True} per run
ArgoUML	4	GP	2	2 (50 %)	0.966
		NSGA-II	3	3 (75 %)	1.9
		SPEA2	2	2 (50 %)	2.0
DPL	1	GP	1	1 (100 %)	0.033
		NSGA-II	1	1 (100 %)	0.166
		SPEA2	1	1 (100 %)	0.066
GOL	8	GP	6	0 (0 %)	0.0
		NSGA-II	8	8 (100 %)	0.966
		SPEA2	6	5 (62 %)	0.933
VOD	1	GP	1	0 (0 %)	0.0
		NSGA-II	1	1 (100 %)	1.0
		SPEA2	1	1 (100 %)	1.0
ZipME	4	GP	1	1 (25 %)	1.0
		NSGA-II	4	4 (100 %)	3.033
		SPEA2	3	3 (75 %)	3.0
MM-V1	1	GP	1	1 (100 %)	0.5
		NSGA-II	1	1 (100 %)	0.5
		SPEA2	1	1 (100 %)	0.566
MM-V2	1	GP	1	1 (100 %)	0.3
		NSGA-II	1	1 (100 %)	0.366
		SPEA2	1	1 (100 %)	0.433
MM-V3	1	GP	2	0 (0 %)	0
		NSGA-II	1	1 (100 %)	0.333
		SPEA2	1	1 (100 %)	0.333
MM-V4	1	GP	1	1 (100 %)	0.066
		NSGA-II	1	1 (100 %)	0.333
		SPEA2	1	1 (100 %)	0.3
MM-V5	1	GP	1	1 (100 %)	0.1
		NSGA-II	1	1 (100 %)	0.233
		SPEA2	1	1 (100 %)	0.166
MM-V6	42	GP	5	1 (2 %)	0.033
		NSGA-II	42	42 (100 %)	4.66
		SPEA2	20	13 (30 %)	1.033
MM-V7	801	GP	5	3 (0.37 %)	0.333
		NSGA-II	796	760 (92 %)	36.0
		SPEA2	155	61 (7 %)	3.833

two values we consider ArgoUML, which has the PF_{True} composed of four solutions. For this case study GP found two solutions that are in PF_{True}, NSGA-II found three solutions, and SPEA2 two. On average GP was able to find almost one solution in PF_{True} per run, what is expected because it is a single-objective approach. On the other hand NSGA-II

found on average 1.9 solutions per run, and SPEA2 2.0 solutions. For this case study GP is able to find good solutions in comparison with the multi-objective algorithms, NSGA-II found the largest amount of solutions in PF_{True} after the thirty runs, but SPEA2 finds more non-dominated solutions on average in each run.

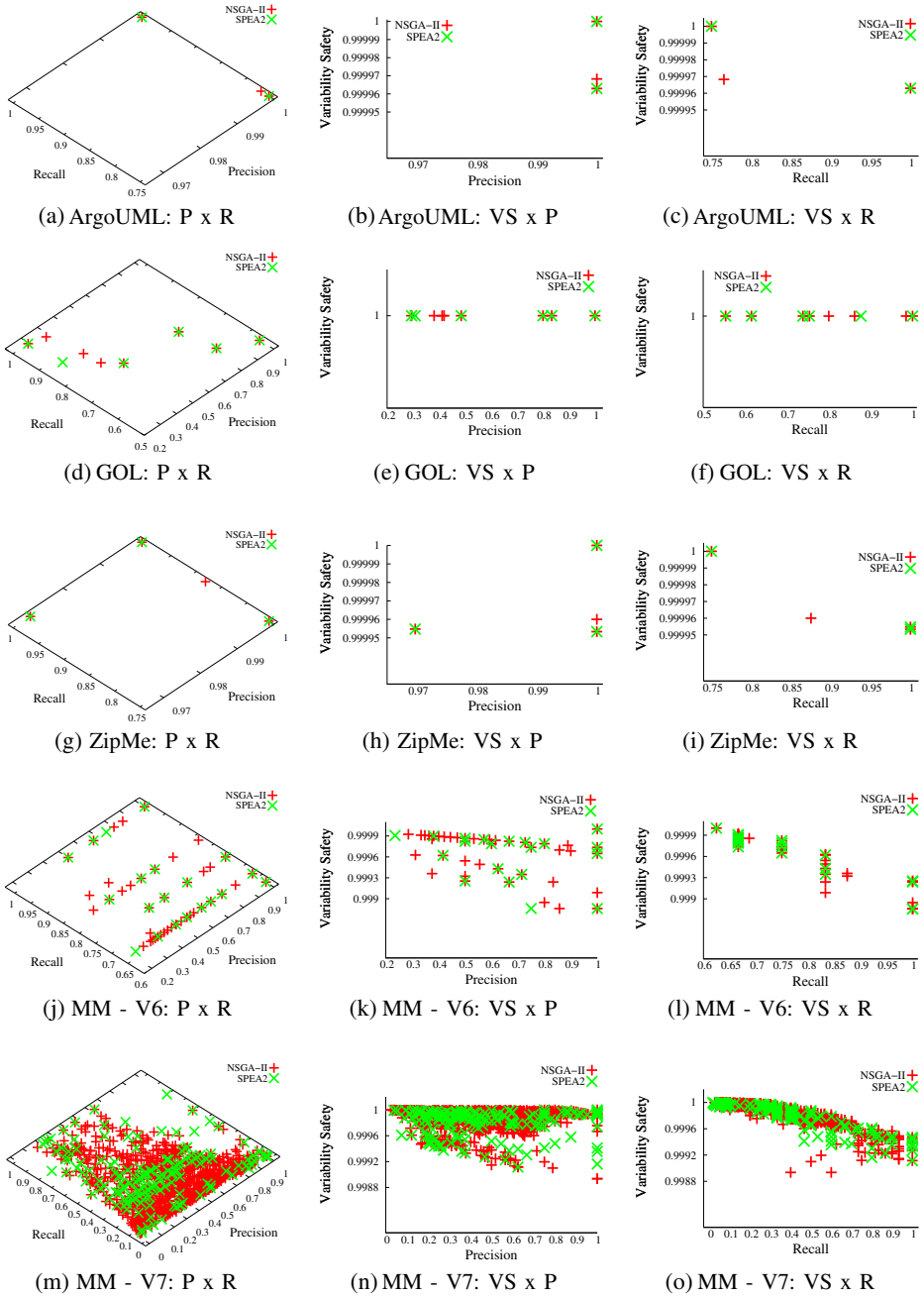
Regarding the number of solutions in PF_{True}, fifth column of Table 6, we can observe that the three algorithms have the same results for five case studies: DPL, MM-V1, MM-V2, MM-V4, and MM-V5. NSGA-II and SPEA2 outperform GP for two case studies: VOD and MM-V3. NSGA-II found more solutions in PF_{True} for five case studies: ArgoUML, GOL, ZipMe, MM-v6, and MM-V7. On the other hand, considering the average of solutions in PF_{True} per run, sixth column of Table 6, the three algorithms have the same results for four case studies: ArgoUML, ZipMe, MM-V1, and MM-v2. NSGA-II and SPEA2 are better than GP in five case studies: GOL, VOD, MM-V3, MM-V4, and MM-V5. NSGA-II outperform GP and SPEA2 in three case studies: DPL, MM-V6, and MM-V7.

In summary, from Table 6 we can observe that GP can only have similar results of NSGA-II and SPEA2 in two case studies and never reached the best results. NSGA-II and SPEA2 reached the same results and outperform GP in four case studies, and in six case studies NSGA-II is better than GP and SPEA2.

RQ1 Discussion From our set of case studies we identified that five of them have conflicting objectives. This means that when we get better values for one objective, then another objective is penalized, leading to a set of possible good solutions with different trade-offs. In such situation the multi-objective algorithms NSGA-II and SPEA2 are better than the single-objective GP. From the results we observed that on average the multi-objective algorithms found a set of solutions per run, on the other hand the single-objective algorithm is able to find only one. Furthermore, considering the set of solutions obtained after the 30 runs, GP still was not competitive against the multi-objective algorithms. GP tends to find in each run the same single solution, not exploring other parts of the search space, which is done by the multi-objective algorithms. For seven case studies the three objectives are not conflicting, however NSGA-II and SPEA2 found the same good solutions found by GP. In summary, we can conclude that a multi-objective approach is the best choice to reverse engineer FMs considering our case studies.

5.2 Answering RQ2

Recall that RQ2 aims at comparing the performance of algorithms NSGA-II and SPEA2 for the three selected objective functions. An interesting characteristic to be analysed is the position of the solutions on the search space. Figure 6 presents the three graphs for each case study with more than one solution in PF_{Known}. The first column of graphs presents the view of objectives precision and recall, the second column the objectives of variability safety and precision, and the third column presents variability safety and recall. As already analysed above, for all these case studies NSGA-II reached a larger amount of solutions than SPEA2. However, in the graphs we can observe that despite the smaller number of solutions, SPEA2 has solutions spread over a very similar area on search space explored by NSGA-II. On the previous analysis of Table 6 we observed that in six case studies NSGA-II reached better solutions than SPEA2, probably because of the larger number of solutions returned by the former algorithm. On the other hand, the smaller amount of solutions reached by SPEA2 can help in situations such as observed in the case studies MM-V6 and MM-V7, where a large number of solutions was returned and the decision maker has to choose only one to be used in practice.



(a) ArgoUML: P x R (b) ArgoUML: VS x P (c) ArgoUML: VS x R

(d) GOL: P x R (e) GOL: VS x P (f) GOL: VS x R

(g) ZipMe: P x R (h) ZipMe: VS x P (i) ZipMe: VS x R

(j) MM - V6: P x R (k) MM - V6: VS x P (l) MM - V6: VS x R

(m) MM - V7: P x R (n) MM - V7: VS x P (o) MM - V7: VS x R

Fig. 6 Solutions on the search space

Statistical Analysis To reason about the differences between NSGA-II and SPEA2 we used the well known quality indicator called Hypervolume (Zitzler et al. 2003). Table 7 presents in the second and third columns the average of Hypervolume and standard

Table 7 Hypervolume and effect size

System	Hypervolume		Wilcoxon	$\hat{A}_{12}$ Effect Size	
	NSGA-II	SPEA2	p-value	NSGA-II	SPEA2
ArgoUML	0.0036 (0.0005)	0.0035 (0.0000)	1.61E-01	46.67	53.33
DPL	0.0064 (0.0025)	0.0071 (0.0017)	1.42E-01	56.56	43.44
GOL	0.0365 (0.0080)	**0.0340 (0.0041)**	**3.62E-02**	34.22	**65.78**
VOD	0.0010 (0.0000)	0.0010 (0.0000)	NA*	50.00	50.00
ZipME	0.0038 (0.0000)	0.0038 (0.0000)	NA*	50.00	50.00
MM-V1	0.0050 (0.0042)	0.0037 (0.0032)	1.69E-01	40.56	59.44
MM-V2	0.0097 (0.0075)	0.0083 (0.0073)	4.33E-01	44.50	55.50
MM-V3	0.0121 (0.0086)	0.0109 (0.0083)	4.33E-01	44.33	55.67
MM-V4	0.0124 (0.0089)	0.0104 (0.0083)	3.49E-01	43.22	56.78
MM-V5	0.0126 (0.0089)	0.0129 (0.0088)	5.06E-01	45.06	54.94
MM-V6	0.0399 (0.0128)	0.0435 (0.0123)	1.78E-01	60.17	39.83
MM-V7	0.1125 (0.0015)	**0.1038 (0.0059)**	**1.86E-09**	4.78	**95.22**

*NA = Not Available, because the two sets of values are identical

deviation, in parentheses, for the 30 runs. To compute the Hypervolume the reference point for all case studies was P=1.1, R=1.1, and VS=1.1. Since our problem is a maximization problem, then lower values of Hypervolume are better. To check statistical difference we applied the Wilcoxon test (Bergmann et al. 2000). The p-value obtained for each case study is presented on the fourth column of Table 7. To corroborate our analysis we also compute the effect size with the Vargha-Delaney's $\hat{A}_{12}$ statistic (Vargha and Delaney 2000), used for assessing randomized algorithms in Software Engineering (Arcuri and Briand 2014), presented on the last two columns of Table 7.

From the results on Table 7 we observe that there is a difference between both algorithms only for the case studies GOL and MM-V7. The boxplots for these two case studies are presented in Fig. 7. In these two systems the best algorithm was SPEA2. The results for the case studies VOD and ZipME are exactly the same. Despite some differences in the average Hypervolume for the other case studies, they are statistically similar.

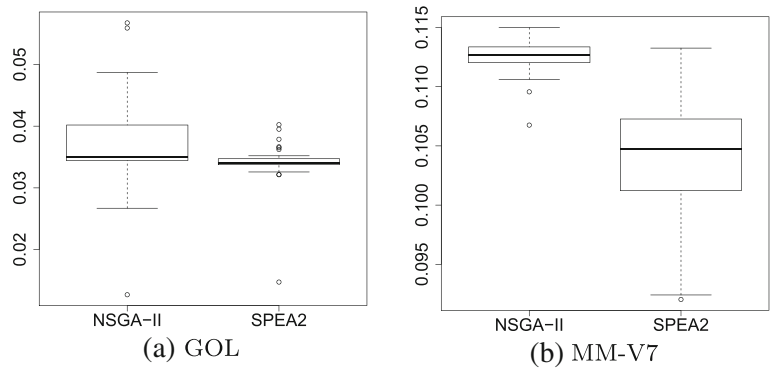


Fig. 7 Hypervolume boxplots

RQ2 Discussion Regarding the number of good solutions found on average and after the 30 runs, NSGA-II outperformed SPEA2. However, both are able to widely explore the search space. Using the well-know Hypervolume indicator we observed that in ten case studies there are no significant differences. In the two case studies with significant difference, the algorithm SPEA2 was the best. In conclusion, both algorithms are good to solve the problem of reverse engineering of FMs using our three objectives.

5.3 Answering RQ3

Recall that the purpose of RQ3 was to explain how our approach could be used in practice. Let us now illustrate how. In Table 8 we present the values of the three objectives for all solutions that compose the sets PF_{known}. The exceptions are case studies MM-V6 and MM-V7. For these two case studies we selected those solutions with the value 1.0 for at least one of the objectives. Here the goal is to analyse qualitatively the solutions reached by the algorithms NSGA-II and SPEA2.

As mentioned before, the solutions of NSGA-II and SPEA2 are the same for seven case studies, namely DPL, VOD, MM-V1, MM-V2, MM-V3, MM-V4, and MM-V5. For the remaining case studies it is possible to observe how conflicting the objectives can be. For example, in ArgoUML we can observe that the conflict occurs only between recall and variability safety, since in all solutions have the value 1.0 for precision. A similar situation happens for GOL, but in this case study the variability safety is 1.0 in all solutions. For the case studies ZipMe, MM-V6 and MM-V7 there are solutions with different trade-offs among the three objectives. These solutions, with different values for each objective, help the software engineers in the decision making. They can decide which measure is more important and then select the corresponding solution.

To illustrate this process of selecting one solution, in Fig. 8 we present four FMs from MM-V7 obtained by SPEA2, their tree-like structure and cross-tree constraints. We selected these FMs based on two criteria: *i)* solutions with value equal to 1.0 for at least one objective, and *ii)* solutions with the best value for a second objective. For example, The solution presented in Fig. 8a has P = 1.0, satisfying the first criteria, and R = 0.8000000119 the best value of recall among solutions with P = 1.0, satisfying the second criteria.

In the FM presented in Fig. 8a we have 192 valid configurations and all of them are desired products, so the value of precision is 1.0. However, the total number of desired products for this case study is 240, so the recall of this FM is not 1.0. The value of variability safety equal to 0.999162264 means that there are 324 broken dependencies from the dependency graph on the valid configurations. The FMs in Fig. 8b and c have the best values for variability safety, however with very low value of recall. In the FM of Fig. 8b only four valid configurations are possible mainly because of the constraint `"Include_CopyMedia EXCLUDES Capture_Photo"` what makes all the configurations after the third level of the tree invalid, because the feature `Include_CopyMedia` is mandatory of `Capture_Photo`. In Fig. 8c only 16 valid configurations are possible, because the constraint `"Include_Video EXCLUDES Include_Music"` makes invalid all the configurations below the second level of the tree, since the feature `Include_Music` is mandatory child of feature `Include_Video`. In Fig. 8d we have an FM with recall equal to 1.0, which means that all desired products are valid configurations here; however, it denotes more valid configurations than the input, hence decreasing the value of precision. In this same FM, the number of broken dependencies is 624. These are four examples of multi-objective solutions that are given to the software engineers, so that they can analyse the trade-offs and select the one that best meets their needs.

Table 8 Non-dominated solutions

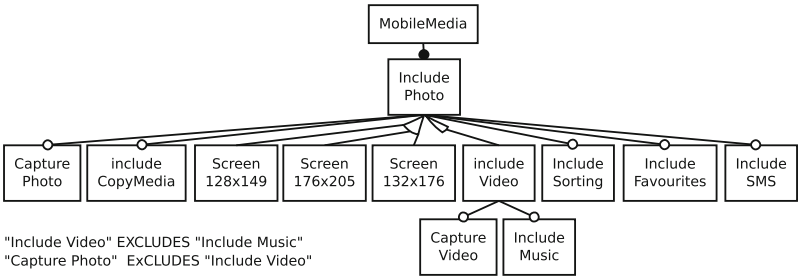
System	NSGA-II			SPEA2		
	P	R	VS	P	R	VS
ArgoUML	1.0	1.0	0.9999629741.0	1.0	1.0	0.9999629741.0
	1.0	0.765625	0.9999682636	1.0	0.75	1.0
	1.0	0.75	1.0			
DPL	1.0	1.0	1.0	1.0	1.0	1.0
GOL	1.0	0.5538461804	1.0	1.0	0.5538461804	1.0
	0.8333333135	0.6153846383	1.0	0.8333333135	0.6153846383	1.0
	0.8000000119	0.7384615541.0	1.0	0.8000000119	0.7384615541.0	1.0
	0.4851485193	0.7538461685	1.0	0.4851485193	0.7538461685	1.0
	0.4193548262	0.8000000119	1.0	0.3081081212	0.8769230843	1.0
	0.4117647111.0	0.8615384698	1.0	0.2941176593	1.0	1.0
	0.380952388	0.9846153855	1.0			
	0.2941176593	1.0	1.0			
VOD	1.0	1.0	1.0	1.0	1.0	1.0
ZipMe	1.0	1.0	0.9999533669	1.0	1.0	0.9999533669
	1.0	0.875	0.9999600288	1.0	0.75	1.0
	1.0	0.75	1.0	0.9696969986	1.0	0.99995478
	0.9696969986	1.0	0.99995478			
MM-V1	1.0	1.0	1.0	1.0	1.0	1.0
MM-V2	1.0	1.0	1.0	1.0	1.0	1.0
MM-V3	1.0	1.0	1.0	1.0	1.0	1.0
MM-V4	1.0	1.0	1.0	1.0	1.0	1.0
MM-V5	1.0	1.0	1.0	1.0	1.0	1.0
MM-V6	1.0	1.0	0.9988592489	1.0	1.0	0.9988592489

Table 8 (continued)

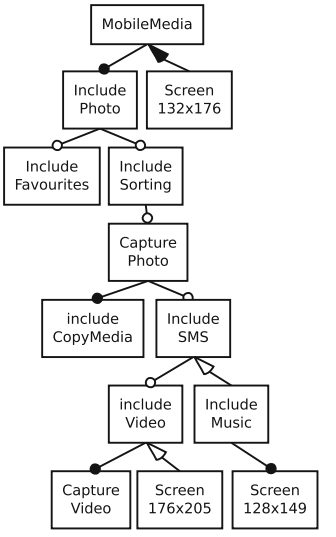
System	NSGA-II			SPEA2		
	P	R	VS	P	R	VS
	1.0	0.8333333135	0.9990873991.0	1.0	0.75	0.9996489996
	1.0	0.75	0.9996489996	1.0	0.6666666865	0.9997367497
	1.0	0.6666666865	0.9997367497	1.0	0.625	1.0
	1.0	0.625	1.0	0.75	1.0	0.9988592489
	0.8571428657	1.0	0.9988592489	0.6666666865	1.0	0.9992394992
	0.8000000119	1.0	0.9989469989	0.5	1.0	0.9992541243
	0.6666666865	1.0	0.9992394992		...	
	0.5	1.0	0.9992541243			
		...				
MM-V7	1.0	0.6000000238	0.9989360141.0	1.0	0.8000000119	0.999162264
	1.0	0.400000006	0.9989360141.0	1.0	0.6000000238	0.9993559012
	1.0	0.3625000119	0.999666002	1.0	0.4666666687	0.9994112262
	1.0	0.2333333343	0.9997554055	1.0	0.400000006	0.9997431758
	1.0	0.1000000015	0.9998858559	1.0	0.200000003	0.9999021622
	1.0	0.0666666701.0	0.9999266217	1.0	0.0833333358	0.9999217298
	1.0	0.0500000007	0.9999347748	1.0	0.0500000007	0.9999347748
	1.0	0.0333333351.0	0.9999510811.0	1.0	0.0333333351.0	0.9999510811.0
	1.0	0.0250000004	1.0	1.0	0.0250000004	0.9999673874
	0.625	1.0	0.9991153834	1.0	0.020833334	0.9999804324
	0.46875	1.0	0.9992723315	1.0	0.0166666675	1.0
	0.375	1.0	0.9993762841.0	0.625	1.0	0.9991153834
	0.25	1.0	0.9994374327	0.46875	1.0	0.9992723315
	0.1666666716	1.0	0.9994700453	0.375	1.0	0.9993347031.0

Table 8 (continued)

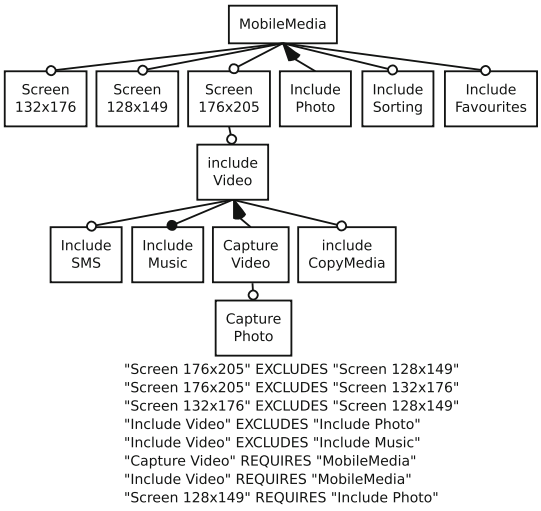
System	NSGA-II			SPEA2		
	P	R	VS	P	R	VS
	0.8000000119	0.0333333351.0	1.0	0.2727272809	1.0	0.9993751723
	0.75	0.0500000007	1.0	0.25	1.0	0.9994374327
		...		0.2307692319	1.0	0.9994393142
				0.2083333284	1.0	0.9994700453
				0.75	0.0500000007	1.0
					...	



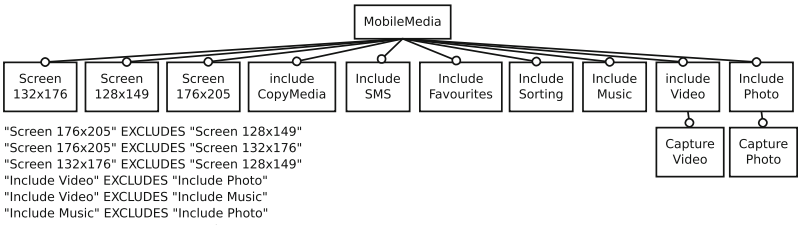
(a) P=1.0, R=0.8000000119, VS=0.999162264



(b) P=1.0, R=0.0166666675, VS=1.0



(c) P=0.75, R=0.0500000007, VS=1.0



(d) P=0.625, R=1.0, VS=0.9991153834

Fig. 8 MM-V7 feature models

The use of dependency graphs to compute variability safety helps to identify another characteristic in the implementation artefacts, the existence of compilation errors or incoherences in the source code. For example, looking at the FM presented in Fig. 8a, the

precision is 1.0, this means that all feature sets denoted by the FM belong to the set of desired feature sets, but there are configurations that even though are valid, they do have broken dependencies. In the dependency graph of this case study there is the dependency $Include\_SMS \Rightarrow Capture\_Photo$, which means that in the source code the feature `Include_SMS` depends on `Capture_Photo`; however, in the set of desired configurations there are 72 configurations that have the feature `Include_SMS` but do not have `Capture_Photo`, decreasing the value of variability safety.

RQ3 Discussion As discussed before, the main advantage of a multi-objective approach is to find a set of good solutions regarding different trade-off among the objectives. In a practical point of view, these solutions allow the software engineer to reason about the different measures used to evaluate the FMs. At the end, the software engineers can select the solution that best fits his/her needs. In addition, multi-objective algorithms can return solutions that can call the attention of the decision maker to characteristics that are not considered in the beginning of the optimization process. We discussed this point about the possible identification of inconsistencies between the source code and the feature sets used as input, e.g. invalid references. To identify such inconsistencies was not the goal of the software engineers when starting the optimization process, however our approach further enables software engineers to reason about inconsistencies by looking at the solutions with different values among the objectives.

5.4 Threats to Validity

The first threat to validity identified was the parameter settings for the algorithms. We addressed this threat by using conventional values for our parameters as we have done in our previous work (Assunção et al. 2015); however, we increased the population size to allow the algorithms to generate more individuals.

The second threat regards to the used case studies. Despite using only twelve case studies, these systems are from different domains and have different sizes. We argue that they are representative to evaluate our approach.

A third threat concerns the baseline comparison. To the best of our knowledge, our approach is the first to use information from implementation artefacts and a multi-objective perspective to reverse engineer feature models. Then as baseline we use the genetic programming algorithm from our previous work (Assunção et al. 2015). Since this single-objective algorithm returns a single solution per run we used averages per run and a set composed with the 30 runs to reason about the differences in comparison with the multi-objective algorithm.

The fourth threat is related to the set of measures we used to evaluate the reverse engineered FMs. Different measures and metrics could produce FMs with other characteristics. However, we believe the three measures we considered are well designed for our goals of representing the actual set of product variants and take into account the source-code structure.

The last threat refers to the validation of the reversed engineered feature models by developers. Certainly, because there are no canonical representations of feature models, domain knowledge can play a significant role when choosing among different feature models. However, this threat is mitigated by considering real case studies that have been extensively studied by us an others for different purposes.

6 Related Work

Search-based techniques have been applied in a wide range of SPL activities such as feature selection, architectural improvement, SPL testing, and feature model construction (Harman et al. 2014; Lopez-Herrejon et al. 2015). To reverse engineer FMs, search-based algorithms were explored in the work of Lopez-Herrejon et al. (2012, 2015). In this work an evolutionary algorithm uses as input a set of desired feature sets and, as objective, a function that maximizes the number of the desired feature sets contained in a feature model disregarding any surplus feature sets that the model could denote (Lopez-Herrejon et al. 2012, 2015). This work was extended by Thianniwet and Cohen Thianniwet and Cohen (2015) to reverse engineer complex features models. The authors designed a fitness function to balance both additional and missing products and create a representation and evolution operators to support complex cross-tree constraints. But none of these works includes the information from the implementation artefacts or a multi-objective perspective.

Feature sets were also used in the work of Haslinger et al. (2011, 2013). The authors used an ad hoc algorithm to identify patterns in the selected and not selected features mapped in parent-child relations of feature models (Haslinger et al. 2011). An extension was done to consider the CTCs requires and excludes (Haslinger et al. 2013). Again the authors do not consider the implementation artefacts.

Czarnecki and Wasowski used a set of propositional logic formulas as input to the reverse engineering task (Czarnecki and Wasowski 2007; She et al. 2014). They propose an ad hoc algorithm to extract multiple feature models from single propositional logic formula preserving the original formulas and reducing redundancy (Czarnecki and Wasowski 2007). Recently, the algorithm has been improved based on CNF and DNF constraints (She et al. 2014). In contrast with our work their starting point are configuration files, documentation files, and constraints expressed in propositional logic instead of feature sets and source code.

Acher et al. proposed an interactive process to map each feature into a feature model, and then all the feature models are merged in a single feature model (Acher et al. 2012). In the work of Sannier et al. they consider product matrices from Wikipedia as start point (Sannier et al. 2013). These matrices can have other values besides select or not select. From these matrices an analysis is performed to identify variability patterns. The authors only mention about the benefits of exploiting that information to extract models such as feature models.

Genetic Programming is also the focus of other pieces of work on software development. An example is the paper of Chan et al. where the authors deal with the problem of product planning and customer satisfaction using a method based on GP (Chan et al. 2011).

There is extensive work using multi-objective optimization algorithms in the field of search-based software engineering. For instance, software module clustering, integration testing, testing resource allocation, protocol tuning, software project scheduling, software project effort estimation, and software defect prediction (Harman et al. 2012; Yao 2013). However, they do not address reverse engineering of FMs.

7 Future Directions

In this section we describe some research opportunities and trends on the task of reverse engineering of feature models.

Note that the variability safety measure is not restricted to implications obtained from a dependency graph. A similar metric can be computed on arbitrary constraints, as long as it can be determined whether they hold or not. This is important because constraints may not

only be provided in the form of a dependency graph as a result of a code analysis tool, but also from other sources like constraints defined by domain experts. Generally speaking this measure expresses the conformance of the found candidate feature models to a set of given constraints.

In this sense, the exploration of different sources of information is a future direction. For instance, the use of test cases is a possible target, since they are usually available in most projects. The comments in the source code are also a possible target for new research. Design models, like UML diagrams, are another interesting starting point to be considered.

Besides the use of different artefacts, another research opportunity is to design new measures and metrics to evaluate the feature models. For example, the use of graph similarity could be applied in dependency graphs of each variant. Perhaps the use of semantic similarity measures among elements is a research opportunity. Non-functional characteristics of systems can be another measure to be included in the reverse engineering process. We also identified a lack of complementary metrics to evaluate and validate the reengineered SPL artefacts, these metrics could facilitate the process of comparison between obtained results and expected ones.

There is a lack of tools that support the reverse engineering of feature model in practice. Recently, tools like BUT4Reuse (Martinez et al. 2015), a framework that provides technologies for leverage commonality and variability of software artefacts, have appeared. However, more tools with different purpose and focus are needed. For example, they should cover different programming languages, different artefact types, etc.

Interactive approaches that include the user in the loop are an alternative strategy to extract information that sometimes is only present in the user's mind. Furthermore, interactive approaches allow the user to guide the reverse engineering process to some preferred directions. Hence performing empirical studies that involve users providing feedback on the generated models is an avenue for future research.

From the identified related work we did not find any strategy to deal with ambiguity in the input artefacts. In other words, the strategies work only with well defined and structured inputs. To deal with incomplete or unsound input is an open challenge.

8 Concluding Remarks

This paper presents an approach to reverse engineer feature models from a set of feature sets and a dependency graph, extracted from the source code. The set of feature sets is used to compute the precision and recall of the feature models. The dependency graph is used to compute the variability safety, i.e. how well-formed the feature model is regarding the implementation artefacts. Furthermore, the approach takes a multi-objective perspective to deal with the three different measures independently, supporting the software engineers in the decision making process.

To evaluate the proposed approach we designed an experiment to answer questions regarding the benefits and advantages of using our multi-objective approach, the performance of two multi-objective evolutionary algorithms for reverse engineering of FMs, and about the practical use of a multi-objective perspective by the software engineers. The experiment was conducted with twelve case studies and used NSGA-II and SPEA2, and a single-objective GP algorithm.

The results indicate that the multi-objective algorithms are better than the single-objective one, which was expected, because in five case studies the three measures are in conflict. From the number of solutions obtained after 30 runs and the average of good

solutions found per run, we observe that the algorithm NSGA-II outperforms the algorithm SPEA2. However, the statistical test using the Hypervolume values indicates they do not have a difference in ten case studies, and in the other two the algorithm SPEA2 has better Hypervolume values.

Reverse engineering of feature models has recently received an increasing attention from the research community on SPLs; however, our work also revealed several research avenues worthy of further investigation. Among them, we can mention: using different types of constraints in addition to the source code dependencies, exploiting new sources of information like non-functional properties, devising new metrics as objective functions, dealing with ambiguity in input artefacts, and developing more robust and interactive tools and techniques.

Acknowledgments This work was supported by Austrian Science Fund (FWF): P 25289-N15, and the Brazilian Agencies CAPES: 007126/ 2014-00 and CNPq: 453678/2014-9 and 305358/2012-0.

References

Acher M, Cleve A, Perrouin G, Heymans P, Vanbeneden C, Collet P, Lahire P (2012) On extracting feature models from product descriptions. In: International workshop on variability modelling of software-intensive systems (vamos), pp 45–54

Arcuri A, Briand L (2014) A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. Software Testing, Verification and Reliability 24(3):219–250. doi:10.1002/stvr.1486

Assunção WK, Lopez-Herrejon RE, Linsbauer L, Vergilio SR, Egyed A (2015) Extracting variability-safe feature models from source code dependencies in system variants. In: Genetic and evolutionary computation conference (GECCO). ACM, New York, NY, USA, pp 1303–1310. doi:10.1145/2739480.2754720

Assunção WKG, Colanzi TE, Vergilio SR, Pozo A (2014) A multi-objective optimization approach for the integration and test order problem. Inf Sci 267:119–139. doi:10.1016/j.ins.2013.12.040

Assunção WKG, Vergilio SR (2014) Feature location for software product line migration: a mapping study. In: 18th software product line conference - 2nd international workshop on REverse variability engineering (REVE), pp 1–8. doi:10.1145/2647908.2655967

Batory DS, Sarvela JN, Rauschmayer A (2004) Scaling step-wise refinement. IEEE Trans Softw Eng 30(6):355–371

Benavides D, Segura S, Cortés AR (2010) Automated analysis of feature models 20 years later: a literature review. Inf Syst 35(6):615–636

Benavides D, Segura S, Trinidad P, Cortés AR (2007) FAMA: tooling a framework for the automated analysis of feature models. In: Pohl K, Heymans P, Kang KC, Metzger A (eds) International workshop on variability modelling of software-intensive systems (VaMoS), Lero Technical Report, vol 2007-01, pp 129–134

Bergmann R, Ludbrook J, Spooren WPJM (2000) Different outcomes of the Wilcoxon-Mann-Whitney test from different statistics packages. Am Stat 54(1):72–77. doi:10.2307/2685616

Chan KY, Kwong CK, Wong TC (2011) Modelling customer satisfaction for product development using genetic programming. J Eng Des 22(1):55–68. doi:10.1080/09544820902911374

Coello CAC, Lamont G, van Veldhuizen D (2007) Evolutionary algorithms for solving multi-objective problems, 2nd edn. Genetic and Evolutionary Computation. Springer, Berlin

Couto MV, Valente MT, Figueiredo E (2011) Extracting software product lines: a case study using conditional compilation. In: Conference on software maintenance and reengineering (CSMR), pp 191–200. doi:10.1109/CSMR.2011.25

Czarnecki K, Wasowski A (2007) Feature diagrams and logics: there and back again. In: International software product line conference (SPLC). IEEE Computer Society, pp 23–34

Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197

Figueiredo E, Cacho N, Sant'Anna C, Monteiro M, Kulesza U, Garcia A, Soares S, Ferrari F, Khan S, Castor Filho F, Dantas F (2008) Evolving software product lines with aspects: an empirical study on design stability. In: International conference on software engineering (ICSE). ACM, New York, NY, USA, pp 261–270. doi:10.1145/1368088.1368124

Fischer S, Linsbauer L, Lopez-Herrejon RE, Egyed A (2014) Enhancing clone-and-own with systematic reuse for developing software variants. In: International conference on software maintenance and evolution (ICSME)

Harman M, Jia Y, Krinke J, Langdon WB, Petke J, Zhang Y (2014) Search based software engineering for software product line engineering: a survey and directions for future work. In: 18Th international software product line conference - volume 1, SPLC '14. ACM, New York, NY, USA, pp 5–18. doi:10.1145/2648511.2648513

Harman M, Mansouri SA, Zhang Y (2012) Search-based software engineering: trends, techniques and applications. ACM Comput Surv 45(1):11:1–11:61. doi:10.1145/2379776.2379787

Haslinger EN, Lopez-Herrejon RE, Egyed A (2011) Reverse engineering feature models from programs' feature sets. In: Working conference on reverse engineering (WCRE), pp 308–312

Haslinger EN, Lopez-Herrejon RE, Egyed A (2013) On extracting feature models from sets of valid feature combinations. In: International conference fundamental approaches to software engineering (FASE), pp 53–67

Kang K, Cohen S, Hess J, Novak W, Peterson A (1990) Feature-Oriented Domain analysis (FODA) feasibility study. Tech. Rep. CMU/SEI-90-TR-21, SEI CMU

van d. Linden FJ, Schmid K, Rommes E (2007) Software product lines in action: the best industrial practice in product line engineering. Springer

Linsbauer L, Lopez-Herrejon RE, Egyed A (2013) Recovering traceability between features and code in product variants. In: International software product line conference (SPLC), pp 131–140

Linsbauer L, Lopez-Herrejon RE, Egyed A (2014) Feature model synthesis with genetic programming. In: International symposium on search based software engineering (SSBSE), pp 153–167

Lopez-Herrejon RE, Galindo JA, Benavides D, Segura S, Egyed A (2012) Reverse engineering feature models with evolutionary algorithms: an exploratory study. In: International symposium on search based software engineering (SSBSE), pp 168–182

Lopez-Herrejon RE, Linsbauer L, Egyed A (2015) A systematic mapping study of search-based software engineering for software product lines. J Inf Softw Technol. doi:10.1016/j.infsof.2015.01.008

Lopez-Herrejon RE, Linsbauer L, Galindo JA, Parejo JA, Benavides D, Segura S, Egyed A (2015) An assessment of search-based techniques for reverse engineering feature models. J Syst Softw 103(0):353–369. doi:10.1016/j.jss.2014.10.037

Manning CD, Raghavan P, Schütze H (2008) Introduction to information retrieval. Cambridge University Press

Martinez J, Ziadi T, Bissyandé TF, Klein J, Traon YL (2015) Bottom-up adoption of software product lines: a generic and extensible approach. In: International conference on software product line (SPLC), pp 101–110. doi:10.1145/2791060.2791086

Sannier N, Acher M, Baudry B (2013) From comparison matrix to variability model: The wikipedia case study. In: International conference on automated software engineering (ASE). IEEE, pp 580–585

Segura S, Galindo J, Benavides D, Parejo JA, Cortés AR (2012) BeTTy: benchmarking and testing on the automated analysis of feature models. In: Eisenecker UW, Apel S, Gnesi S (eds) International workshop on variability modelling of software-intensive systems (VaMoS). ACM, pp 63–71

She S, Lotufo R, Berger T, Wasowski A, Czarnecki K (2011) Reverse engineering feature models. In: International conference on software engineering (ICSE). ACM, pp 461–470

She S, Ryssel U, Andersen N, Wasowski A, Czarnecki K (2014) Efficient synthesis of feature models. Inf Softw Technol 56(9):1122–1143

Thianniwet T, Cohen M (2015) Splrevo: optimizing complex feature models in search based reverse engineering of software product lines. In: North american search based software engineering symposium (NasBASE)

Vargha A, Delaney H (2000) A critique and improvement of the cl common language effect size statistics of mcgraw and wong. J Educ Behav Stat 25(2):101–132

Weston N, Chitchyan R, Rashid A (2009) A framework for constructing semantically composable feature models from natural language requirements. In: International software product line conference (SPLC), pp 211–220

Yao X (2013) Some recent work on multi-objective approaches to search-based software engineering. Springer, Berlin, pp 4–15. doi:10.1007/978-3-642-39742-4_2

Zitzler E, Laumanns M, Thiele L (2001) SPEA2: improving the strength pareto evolutionary algorithm. Tech. Rep. 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland

Zitzler E, Thiele L, Laumanns M, Fonseca CM, da Fonseca VG (2003) Performance assessment of multiobjective optimizers: an analysis and review. IEEE Trans Evol Comput 7:117–132

Wesley K. G. Assunção received a bachelor's degree in Information Systems from Faculdade Sul Brasil in 2006 and the MSc degree in 2012 from Federal University of Paraná (UFPR), Brazil. He is currently PhD candidate at the Post-graduation Program in Informatics of Federal University of Paraná (UFPR), being a member of Research Group on Software Engineering - GRES. His areas of interest are: Software Testing, Software Product Lines, Search Based Software Engineering and Multi-Objective Evolutionary Algorithms.

Roberto E. Lopez-Herrejon is an Associate Professor at the Department of Software Engineering and Information Technology of the École de Technologie Supérieure of the University of Quebec in Montreal, Canada. Prior he was a senior postdoctoral researcher at the Johannes Kepler University in Linz, Austria. He was an Austrian Science Fund (FWF) Lise Meitner Fellow (2012–2014) at the same institution. From 2008 to 2014 he was an External Lecturer at the Software Engineering Masters Programme of the University of Oxford, England. From 2010 to 2012 he held an FP7 Intra-European Marie Curie Fellowship sponsored by the European Commission. He obtained his Ph.D. from the University of Texas at Austin in 2006, funded in part by a Fulbright Fellowship sponsored by the U.S. State Department. From 2005 to 2008, he was a Career Development Fellow at the Software Engineering Centre of the University of Oxford sponsored by Higher Education Founding Council of England (HEFCE). His main expertise is in software customization, software product lines, and search based software engineering.

Lukas Linsbauer is currently a PhD student at the Institute for Software Systems Engineering at the Johannes Kepler University (JKU) in Linz, Austria under the supervision of Prof. Alexander Egyed and Dr. Roberto Erick Lopez-Herrejon. He received his master's degree in computer science from the JKU after only four years of study for each of which he received a merit scholarship. His research interests are in traceability, software product lines, variability modeling and management, and highly variable and configurable systems.

Silvia R. Vergilio received the MS (1991) and DS (1997) degrees from University of Campinas, UNICAMP, Brazil. She is currently at the Computer Science Department at the Federal University of Paraná, Brazil, where she has been a faculty member since 1993. She has been involved in several projects and her research interests are in the areas of Software Engineering, such as: software testing, software quality and software metrics.

Alexander Egyed heads the Institute for Software Systems Engineering (ISSE) at the Johannes Kepler University, Austria. He received his Doctorate from the University of Southern California, USA and previously worked many years in industry before joining academia. Dr. Egyed was recognized among the Top 10 scholars in software engineering and his work has received numerous awards.

Appendix C

Discovering Software Architectures with Search-based Merge of UML Model Variants

Discovering Software Architectures with Search-based Merge of UML Model Variants

Wesley K.G. Assunção[1,2], Silvia R. Vergilio[2*], and Roberto E. Lopez-Herrejon[3]

[1]DInf - Federal University of Paraná, CP: 19081, CEP 81531-980, Curitiba, Brazil
[2]COINF - Federal University of Technology - Paraná, CEP: 85902-490, Toledo, Brazil
[3]ÉTS - University of Quebec, Notre-Dame Ouest 1100, H3C 1K3, Montreal, Canada
{wesleyk,silvia}@inf.ufpr.br,roberto.lopez@etsmtl.ca

Abstract. Software reuse is a way to reduce costs and improve quality. However, in industry, the reuse of existing software artifacts is commonly done by ad hoc strategies such as clone-and-own. Clone-and-own leads to a set of system variants developed independently, despite of having similar parts. The maintenance of these independent variants is a difficult task, because of duplication and spread functionalities. One problem faced by developers and engineers is the lack of a global view of such variants, providing a better understanding of the actual state of the systems. In this paper we present an approach to discover the architecture of system variants using a search-based technique. Our approach identifies differences between models and uses these differences to generate candidate architectures. The goal is to find a candidate architecture most similar to a set of UML model variants. Our contribution is threefold: i) we proposed an approach to discover model-based software architecture, ii) we deal with the merging of multiple UML model variants; and iii) our approach applies a search-based technique considering state-based merging of models. We evaluate our approach with four case studies and the results show that it is able to find good candidate architectures even when different features are spread among model variants.

Keywords: Model merging; UML models; Model-based architectures; Search-based techniques

1 Introduction

Developing software systems from scratch is a complex and high cost activity. A well established strategy to reduce costs, improve productivity and increase quality is software reuse, which is based on the use of existing artifacts to develop new software systems [14]. Any artifact built during software development can be reused, including source code, design models, test cases, etc.

Software reuse is generally carried out using an ad hoc strategy, called "clone-and-own" [20]. In this strategy, existing software artifacts are cloned/copied and

* This work was supported by the Brazilian Agencies CAPES: 7126/2014-00 and CNPq: 453678/2014-9 and 305358/2012-0.

adapted to fulfill the new requirements. The clone-and-own strategy is an easy way to reuse software, does not require an upfront investment, while obtaining good results quickly. However, the simultaneous maintenance and evolution of a typically large number of individual system variants is a complex activity because different variants can provide the same functionalities, but at the same time modify or add others. These duplicated functionalities must be maintained individually [6]. Furthermore, engineers commonly do not have a global view on how the different implementations are spread over the variants.

There exists extensive work on migration of multiple variants into SPLs [1]. However, this task demands high levels of investment and human resources [19]. There is a lack of effective ways to support the maintenance and evolution of multiple variants. One way to deal with this problem is the creation of a documented architecture. Software architectures are artifacts that provide a high-level view of functional parts of systems and allow analysis of their structure [4]. An architecture supports design decisions and eases software reuse. Currently, majority of the work on software architecture recovery/discovery is based on source code [7, 10, 11].

In this paper, we present an approach to automatically merge multiple UML model variants to obtain a documented software architecture. The goal is to discover a global model that contains all the features spread across the different variants. A *feature* is a user-visible aspect, functionality, or characteristic of a system [12]. The input of our approach is a set of UML model variants and the output is a complete model, the most similar to all variants. The proposed merging process relies on a search-based technique to avoid having to deal with domain specific constraints of systems under consideration and possible conflicts among models. In other words, we delegate to the evolutionary process the solution of problems regarding constraints and conflicts [9]. We implemented our approach with a genetic algorithm, and evaluated it using four case studies from different domains and with different sizes. Our evaluation showed that good candidate architectures can be found.

The main contributions of our work are:

- Our study relies on the discovery of model-based architectures from different UML model variants. In the literature there are few pieces of work with focus on discovery of architectures from diagrams/models [15, 17].
- The proposed search-based merging approach deals with multiple UML models variants, whereas the majority of studies on model merging considers only two or three models at once [3, 13, 16].
- Our approach performs state-based merging of models, i.e. we consider the model itself during the evolutionary process. Other pieces of work that merge models are operation-based, which means they work mainly considering the history of operations applied to the different models created [13, 16].

The remainder of this paper is as follows. In Section 2 we describe in detail the proposed search-based approach. The evaluation of the proposed approach and the results are presented in Section 3. Section 4 reviews related work. Section 5 contains conclusions and future work.

2 Proposed Approach

In this section we present details of our search-based approach to discover software architectures. According to Harman et al. [9], three ingredients are necessary to implement a search-based approach: (i) an appropriate way to represent solutions, (ii) a function designed to evaluate the quality of a solution, and (iii) a set of operators to generate new solutions and explore the search space.

In this section we describe such ingredients of our approach. To illustrate how it works, we rely on three variants of a Banking System [17]. These variants[1] are presented in Figure 1. The goal of our approach is to obtain a global UML model with the greatest similarity to the variants of our example. To reach this greatest similarity, the global UML model must have as many as possible of the features contained across the variants.

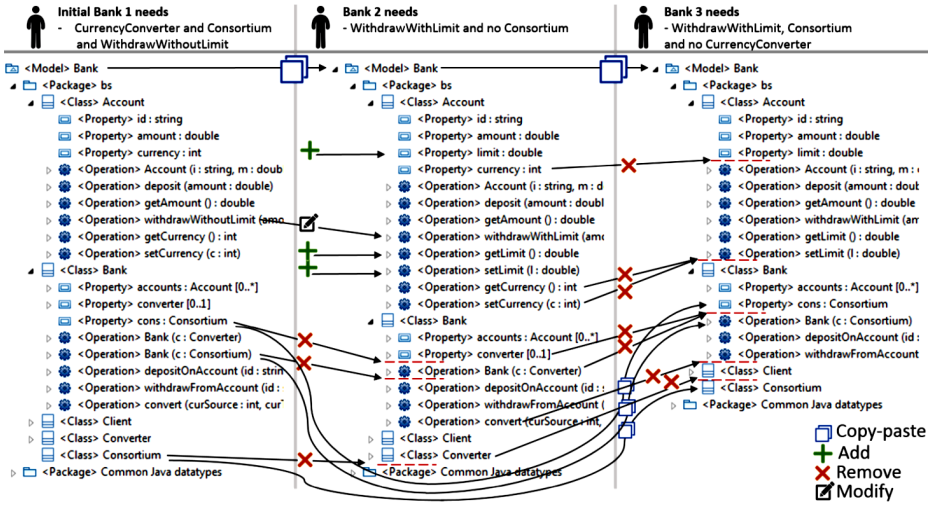


Fig. 1. Three Banking System model variants, extracted from [17]

2.1 Representation

Our approach deals with models created with the Eclipse Modeling Framework (EMF). EMF is a well-known and widely used set of modeling tools [22]. We represent the models using EMF-based UML2[2] implementation of the UML^{TM} 2.x metamodel for the Eclipse platform. When models are represented using EMF-based UML2 data types, they can be compared, modified, and saved. These operations enabled by EMF tools are the basis to our search-based approach.

[1] Available at: `https://github.com/but4reuse/but4reuse/wiki/Examples`
[2] `http://wiki.eclipse.org/MDT/UML2`

2.2 Fitness Function

The fitness function of our approach is based on differences among UML models of system variants. To compute these differences we use the Eclipse EMF Diff/Merge tool[3]. EMF Diff/Merge compares two models and returns the differences between them. EMF Diff/Merge computes three essential types of differences between models: (i) presence of an unmatched element, which refers to an element in a model that has no match in the opposite model; (ii) presence of an unmatched reference value, which means that a matched element references another element in only one model; (iii) presence of an unmatched attribute value, where a matched element owns a certain attribute value in only one model.

Figure 2 presents the output of EMF Diff/Merge when comparing differences between the variants Bank 1 and Bank 2 (Figure 1). The total number of differences is thirteen, but it is composed of two sets of differences. At the top of the figure we have seven differences that are elements present in Bank 2 but missing on Bank 1, and at the bottom of the figure we have six elements that belong to Bank 1 but do not appear in Bank 2.

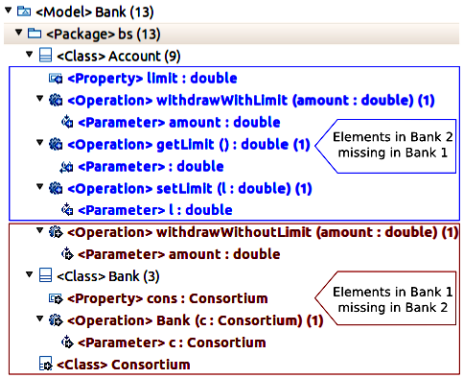


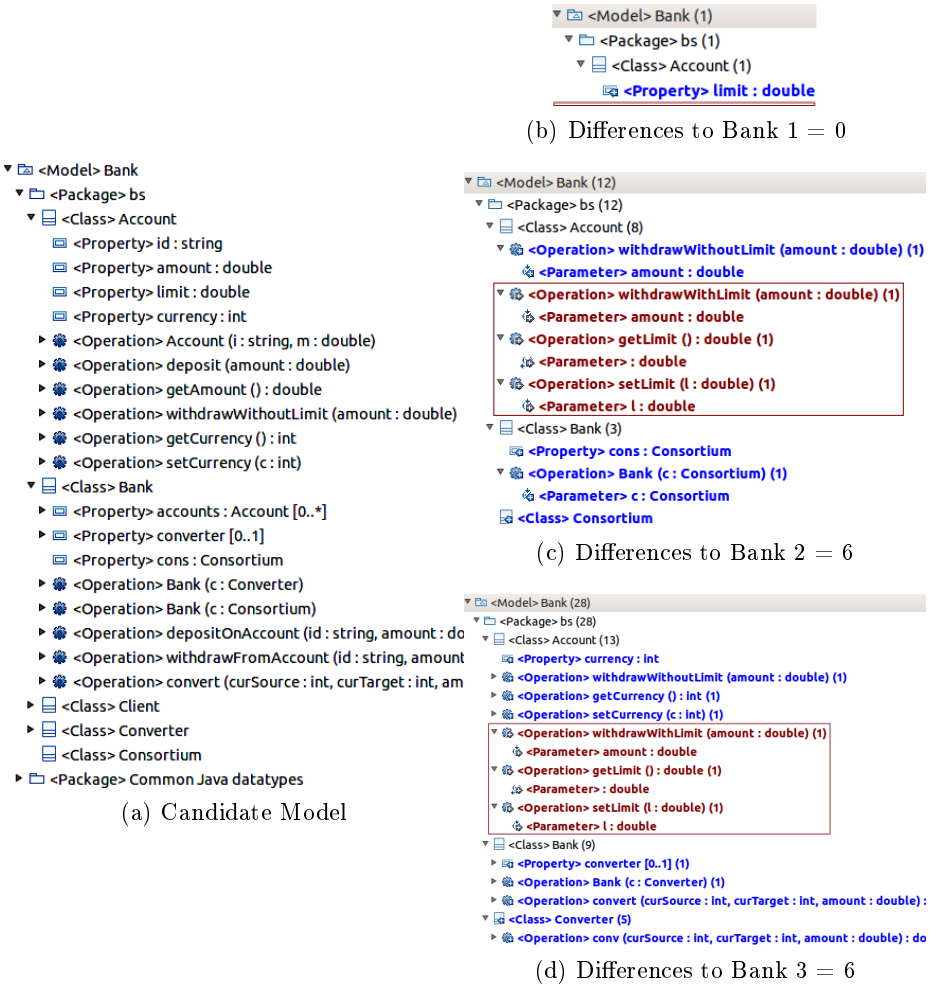
Fig. 2. Differences between variants Bank 1 and Bank 2

EMF Diff/Merge tool is able to compare only two or three models at once. However, to evaluate a candidate architecture we have to compute differences from one model to many model variants. Considering this, the proposed fitness function is composed of the sum of differences from one model to all input model variants. Definition 1 presents the fitness function called here *Model Similarity*. The function *diff* represents the number of differences found by using EMF Diff/Merge, but here we sum only the set of differences that indicate the elements that exist in the variant v but are missing on the $candidate\_model$. There are no distinctions among the three essential types of differences.

[3] `http://www.eclipse.org/diffmerge/`

Definition 1. ***Model Similarity (MS).*** *Model Similarity expresses the degree of similarity of the candidate architecture model to a set of model variants.*

$$MS = \sum_{v \in Variants} diff(candidate\_model, v) \quad (1)$$



(a) Candidate Model

(b) Differences to Bank 1 = 0

(c) Differences to Bank 2 = 6

(d) Differences to Bank 3 = 6

Fig. 3. Example of fitness evaluation

To illustrate the computation of MS we consider the candidate architecture model presented in Figure 3(a) and the input models in Figure 1. Using EMF Diff/Merge tool we have obtained the sets of differences presented in Figures 3(b), 3(c) and 3(d), respectively to Bank 1, Bank 2 and Bank 3. For our fitness function only the differences from the candidate architecture to each variant are relevant, they are highlighted in the figures. There are no differences from the candidate

model to Bank 1. From candidate model to Bank 2 there exist six differences. From candidate model to Bank 3 we have also six differences. We can observe twelve differences from the candidate model to all input variants, then MS = 12. The goal is to minimize the value of MS. An ideal solution has MS equal to zero, which indicates that the candidate architecture has all elements from the variants for which we want to discover the corresponding architecture.

2.3 Genetic Operators

The set of differences returned by EMF Diff/Merge is used to perform crossover and mutation and it also allows duplication/modification of models to incorporate the changes done by the operators.

Crossover The start point of our crossover operator is two candidate architectures. From these two models we generate two children: one with the differences merged and one without the differences. For instance, let us consider any parent models X and Y. The children will be:

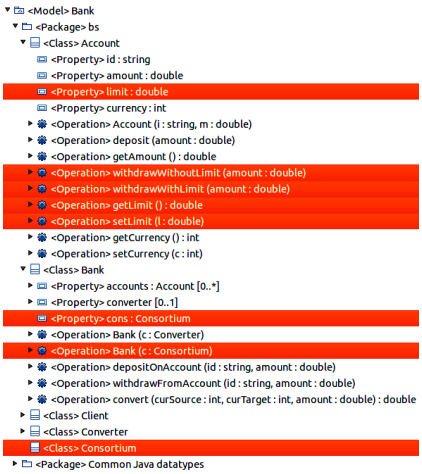
- *Crossover Child Model 1*: this model has the differences between its parents merged. For example, the elements of X that are missing on Y are merged in this later, or vice versa. Both ways will produce the same child.
- *Crossover Child Model 2*: this child is generated by removing the differences between the parents. For example, the differences of X that are missing on Y are removed, or vice versa. Both ways will produce the same child.

The strategy adopted by child model 1 aims at creating a model that has more elements, going to the direction of the system architecture. On the other hand, the strategy used by child model 2 has the goal of eliminating possible conflicting elements from a candidate architecture.

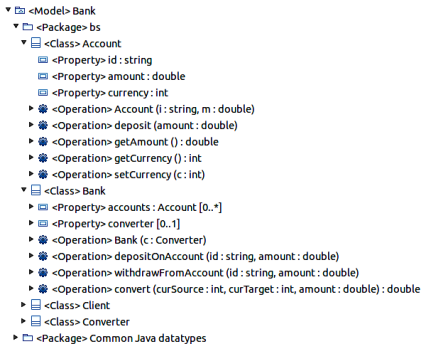
To illustrate the crossover operator let us consider as parents Bank 1 and Bank 2 presented in Figure 1 and the differences between them, presented in Figure 2. The offspring generated by crossover is presented in Figure 4. In Figure 4(a) we have the child with all differences merged (highlighted) and in Figure 4(b) the child with the differences removed.

Mutation Mutation operator aims at applying only one modification in each model parent. The start point of mutation operator is two candidate architectures, and the result is also two children. Let us again consider any parent models X and Y. The children are:

- *Mutation Child Model 1*: the first child is created by merging one difference of the model Y in the model X. After randomly selecting one element of the model Y, but missing on the model X, this element is added in the model X.
- *Mutation Child Model 2*: the same process described above is performed but including one element of the model X in the model Y.



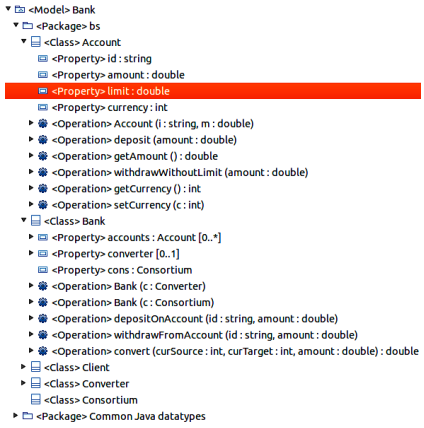
(a) Crossover Child 1



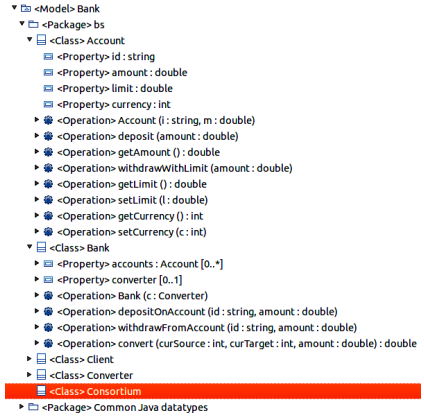
(b) Crossover Child 2

Fig. 4. Example of crossover between Bank 1 and Bank 2

An example of mutation between Bank 1 and Bank 2 (Figure 1) is presented in Figure 5. Considering the differences shown in Figure 2, we have have seven differences to select one to include in Bank 1, and six differences to select one to include in Bank 2. As highlighted in Figure 5(a), the attribute `limit` was chosen to be included in Bank 1. In the child of Figure 5(b) we can see that the class `Consortium` was selected to be included in Bank 2.



(a) Mutation Child 1



(b) Mutation Child 2

Fig. 5. Example of mutation between Bank 1 and Bank 2

The mutation process can select a difference that is owned (i.e. is part of) another difference. In such cases, the entire owning difference is moved to the child. For example, when a mutation selects a parameter owned by an operation, the entire operation is moved to the child.

Selection We use binary tournament strategy whereby a set of individuals are randomly selected from the population, from which the individual with the best fitness is chosen to undergo crossover and mutation [8].

Initial Population The initial population is created by copying the input UML models variants. All variants should be included in the initial population at least once. Each copied variant is an individual. More than one copy of each variant is allowed to reach the population size.

3 Evaluation

In this section we present the setup and case studies used to evaluate the proposed approach, along with the results obtained and their analysis.

3.1 Implementation Aspects and Experimental Setup

We implemented our work using JMetal framework which provides several algorithms for multi-objective and mono-objective optimization [5]. We selected the mono-objective generational Genetic Algorithm (GA) [8]. Our GA was designed to deal with an minimization problem, recall that an ideal solution for our architecture recovery problem is an individual (i.e. candidate architecture) with fitness equal to zero (0).

As mentioned before, in our implementation we handle the UML models using EMF framework. This framework was used mainly to load and save models. For the evolutionary process, where we compare and modify models, we used EMF Diff/Merge. Despite of EMF Diff/Merge having many functionalities, we needed to develop a customized match policy. The default match policies of EMF Diff/Merge only perform comparisons based on XMI:ids. However, model variants could have similar semantic even with different structures. Our customized match police considers qualified names, data types, and relationship types.

The GA parameters were: population size = 200, crossover probability = 0.95, mutation probability = 0.2, and number of fitness evaluations = 8000. We have set the parameters of crossover and mutation based on default values used in other discrete problems on JMetal. Population size and number of evaluations were set based on hardware limitation. When we tried to use greater values for these two latter parameters it caused limit memory exception. The elitism strategy for the generation GA was copying the best four individuals of one generation for the next generation. The number of fitness evaluations is the stop criteria. The experiments were run on a machine with an Intel® Core^{TM} i7-4900MQ CPU with 2.80 GHz, 16 GB of memory, and running a Linux platform.

3.2 Case Studies

In our experiment we used four case studies. Each case study is a set of different UML model variants where each variant implements different system features, and is composed of classes, attributes, operations and relationships. The case studies are: Banking System (BS), a small banking application composed of four features [18]; Draw Product Line (DPL), a system to draw lines and rectangles with six features [2]; Video On Demand (VOD) implements eleven features for video-on-demand streaming [2]; and ZipMe (ZM), a set of tools to files compression with seven features [2]. The variants are presented in Tables 1 to 4, respectively. These tables show the features, number of classes (#Cl), number of attributes (#Attr), number of operations (#Op), and number of relationships (#Rel) for each variant. This information was computed using SDMetrics[4]. Only BS is originally a set of UML model variants, for the other case studies we reverse engineered the models from Java code using the Eclipse MoDisco[5].

For the four case studies we have variants with all possible features combinations. However, we selected only variants that implement at most half of the non-mandatory features. To select these variants we follow the rule:

$$threshold = (RoundUp\left(\frac{\#all\_features - \#mandatory\_features}{2}\right) + \#mandatory\_features)$$

We selected for our experiment only variants that implement a number of features below the threshold. The reason to select only a sub-set of variants is to have the combinations of features spread on different variants, to assess the ability of our approach to merge the models and get good system architectures. For each case study we also had a variant that implements all features, i.e. the most complete variants. We use this variant as a baseline for our analysis, since we consider this variant as the most similar model to a known system architecture. In the last line of Tables 1 to 4 there is information about the baseline.

Table 1. Banking System

Variant	Features				#Cl	#Attr	#Op	#Rel
	BS	WL	CON	CC				
1	✓				3	5	6	1
2	✓		✓		4	6	7	3
3	✓	✓			3	6	8	1
4	✓			✓	4	7	11	2
Baseline	✓	✓	✓	✓	5	9	14	4

BS: Base, WL: Withdraw Limit, CON: Consortium, CC: Currency Converter

Observing the information in case studies tables (Tables 1 to 4) we can see that there are no variants with as many features as the baselines. Furthermore, the number of classes, attributes, operations, and relationships in the variants of all case studies are smaller than the baselines.

[4] http://www.sdmetrics.com

[5] https://eclipse.org/MoDisco

Table 2. Draw Product Line

Variant	Features						#Cl	#Attr	#Op	#Rel
	DPL	L	R	C	W	F				
1	✓	✓					4	13	26	3
2	✓	✓	✓				5	24	37	4
3	✓		✓				4	18	29	3
4	✓	✓		✓			4	22	27	3
5	✓		✓	✓			4	27	30	3
6	✓	✓			✓		4	15	27	3
7	✓		✓		✓		4	20	30	3
8	✓		✓	✓		✓	4	33	32	3
Baseline	✓	✓	✓	✓	✓	✓	5	42	41	4

DPL: Base, L: Line, R: Rectangle, C: Color, W: Wipe, F: Fill

Table 3. Video On Demand

Variant	Features											#Cl	#Attr	#Op	#Rel
	VOD	SP	SelM	StaM	PI	VRC	P	StoM	QP	CS	D				
1	✓	✓	✓	✓	✓	✓						32	362	217	75
2	✓	✓	✓	✓	✓	✓	✓					32	362	217	75
3	✓	✓	✓	✓	✓	✓		✓				33	364	221	77
4	✓	✓	✓	✓	✓	✓	✓	✓				33	364	221	77
5	✓	✓	✓	✓	✓	✓			✓			33	364	221	77
6	✓	✓	✓	✓	✓	✓	✓		✓			33	364	221	77
7	✓	✓	✓	✓	✓	✓		✓	✓			34	366	225	79
8	✓	✓	✓	✓	✓	✓				✓		37	377	232	87
9	✓	✓	✓	✓	✓	✓	✓			✓		37	377	232	87
10	✓	✓	✓	✓	✓	✓		✓		✓		38	379	236	89
11	✓	✓	✓	✓	✓	✓			✓	✓		38	379	236	89
12	✓	✓	✓	✓	✓	✓					✓	35	374	226	82
13	✓	✓	✓	✓	✓	✓	✓				✓	35	374	226	82
14	✓	✓	✓	✓	✓	✓		✓			✓	36	376	230	84
15	✓	✓	✓	✓	✓	✓			✓		✓	36	376	230	84
16	✓	✓	✓	✓	✓	✓				✓	✓	40	389	241	94
Baseline	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	42	393	249	98

VOD: Base, SP: Start Player, SelM: Select Movie, StaM: Start Movie, PI: Play Imm, VRC: VRC Interface, P: Pause, StoM: Stop Movie, QP: Quit Player, CS: Change Server, D: Details

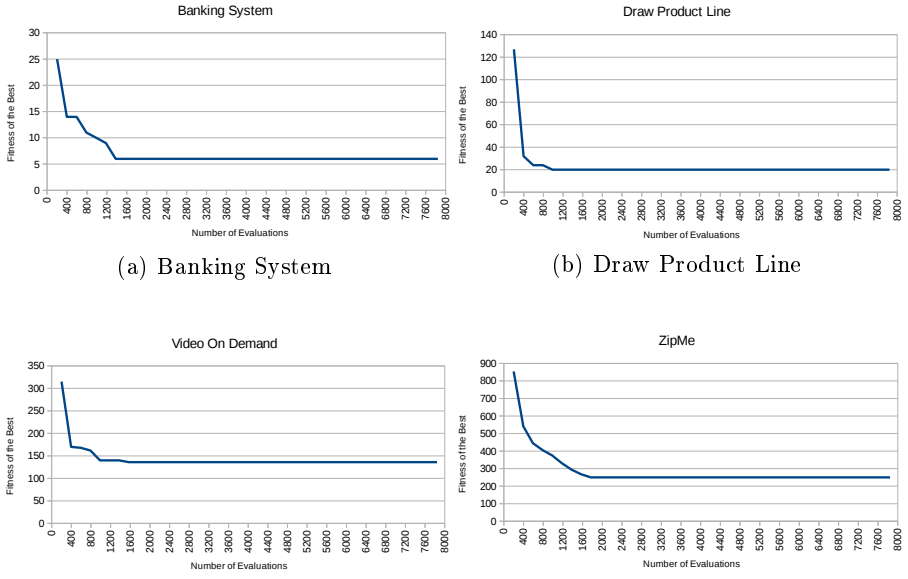
Table 4. ZipMe

Variant	Features							#Cl	#Attr	#Op	#Rel
	ZM	C	CRC	AC	GZIP	A32	E				
1	✓	✓						22	212	241	64
2	✓	✓	✓					23	215	251	66
3	✓	✓		✓				22	212	243	66
4	✓	✓	✓	✓				23	215	253	68
5	✓	✓			✓			25	223	263	68
6	✓	✓	✓		✓			26	229	282	72
7	✓	✓		✓	✓			25	223	265	70
8	✓	✓				✓		23	216	263	69
9	✓	✓	✓			✓		24	219	273	71
10	✓	✓		✓		✓		23	216	265	71
11	✓	✓			✓	✓		26	227	285	73
12	✓	✓					✓	23	219	262	70
13	✓	✓	✓				✓	24	223	279	74
14	✓	✓		✓			✓	23	219	264	72
15	✓	✓			✓		✓	26	230	284	74
16	✓	✓				✓	✓	24	223	284	75
Baseline	✓	✓	✓	✓	✓	✓	✓	28	241	334	87

ZM: ZipMe, C: Compress, CRC: CRC-32 checksum, AC: Archive Check, GZIP: GZIP format support, A32: Adler32 checksum, E: Extract

3.3 Results and Analysis

Figure 6 shows the evolution of the best candidate architecture in each GA generation. As mentioned before, the initial population is composed of copied the input models. The best individual of each case study after the first 200 fitness evaluations is an input model from the initial population that has the least difference from the other input models. For BS the best individual is Variant 4 that has 25 differences from the input. For DPL the best initial individual is Variant 2 with 127 differences. For VOD the best initial candidate architecture is Variant 16 with 315 differences. Finally, Variant 11 of ZM is the best individual of the initial population having 854 differences from the input. These individuals are the first solutions presented in the charts of Figure 6. Observing the figures we can see how the evolutionary process is able to find better candidate architectures by reducing the number of differences. On average the best solution is found after 1400 fitness evaluations. VOD is the simplest case study, since the best solution was reached with approximately 1000 fitness evaluations. On the other hand, ZM is the most complex case study, needing approximately 1800 fitness evaluations to reach the best solution. As expected for a GA, in all case studies there is a great improvement in the number of found solutions in the initial generations, and then the search remains stable.



(a) Banking System

(b) Draw Product Line

(c) Video On Demand

(d) ZipMe

Fig. 6. Evolution of the Best Individual

Another information gathered during the experimentation is the runtime. The amount of time spent by the GA to perform the entire evolutionary process

was: BS = 55s 740ms, DPL = 6m 13s 17ms, VOD = 1h 46m 55s 698ms, and ZM = 2h 10m 29s 267ms. GA ran very fast for BS, that has the smallest number of features, classes, attributes, operations, and relationships. DPL has more features and model elements (Table 2) than BS, and for this case study, the GA took a little more than 6 minutes. A huge difference on the runtime is observed for VOD and ZM. VOD needed almost 2 hours to be finished. ZM is the case study which required the biggest amount of time, it took more than 2 hours.

Now let us consider the details of the best solutions found. Table 5 shows the information of candidate architectures and baseline models. The values of MS presented in the third column is in relation to the input models. Regarding the number of classes, attributes, operations and relationships, the baseline model and the best individual model are very similar. For BS there is only a single difference in the number of relationships, where the best individual has one relationship less. In DPL and VOD the number of model elements are the same. For ZM the number of model elements is different in operations and relationships. Despite having a similar number of model elements, we can observe that the values of MS are not similar. As mentioned before in Section 2.2, the fitness function EMF Diff/Merge computes the presence of elements, presence of attributes values, and presence of reference values. This latter difference happens when a model element references to, or belongs to, different model elements. This explains the reason why baselines and best individuals have similar number of model elements but different values of MS.

Table 5. Candidate Architectures

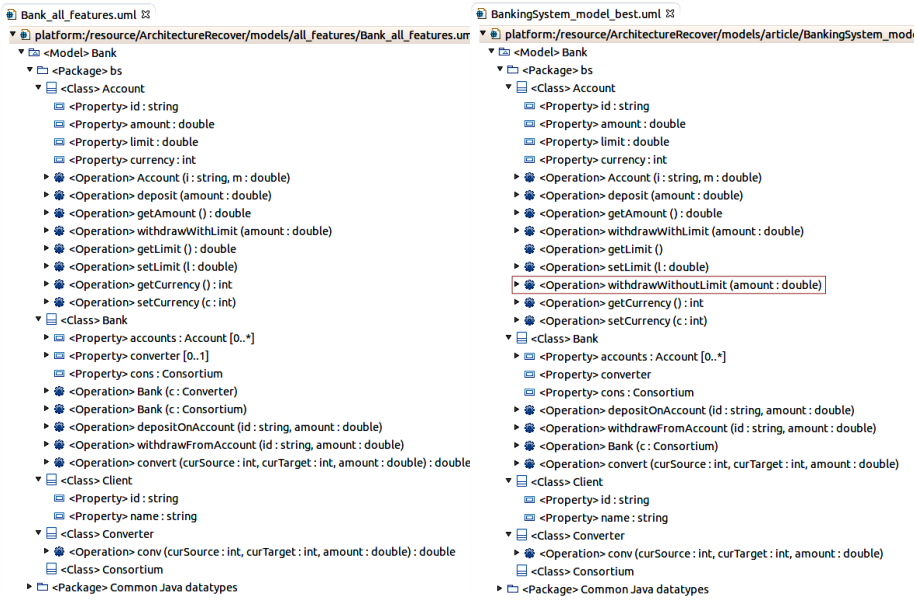
Case Study	Model	MS	#Cl	#Attr	#Op	#Rel
BS	Baseline	20	5	9	14	4
	Best Individual	6	5	9	14	3
DPL	Baseline	40	5	42	41	4
	Best Individual	20	5	42	41	4
VOD	Baseline	162	42	393	249	98
	Best Individual	136	42	393	249	98
ZM	Baseline	633	28	241	334	87
	Best Individual	250	28	241	381	79

#Cl: Number of classes, #Attr: Number of attributes,
#Op: Number of operations, #Rel: Number of relationships

Table 6 presents the differences between baseline and the best individuals for each case study. Since the comparison of EMF Diff/merge has two directions, we show the number of differences existing from baseline to the best individual (candidate architecture), and vice versa. For example, considering BS, there are seven differences needed for baseline having all elements of the candidate architecture. On the other hand, candidate architecture needs fourteen existing differences to have all elements of baseline. In the values of Table 6 we can observe that the baseline is the less different for the case studies BS, DPL and VOD. This means that it is easier to transform baseline in the best than vice versa. For ZM, the solution obtained by the GA is the most similar to the baseline.

Table 6. Differences Between Baseline and Candidate Architectures

Case Study	From Baseline To Best	From Best To Baseline
BS	7	14
DPL	5	451
VOD	20	3425
ZM	4155	200



(a) Baseline (b) Best Solution

Fig. 7. Baseline and Best Solution for Banking System

The analysis of Tables 5 and 6 reveals that a model having all features does not imply that it is the most similar to a set of model variants. We can infer this by considering that the best individual obtained by the GA for each case study is the most similar to the model variants than the baseline (third column in Table 5), and on the other hand baseline is more similar to the best individual when comparing these two models (second and third columns of Table 6). To illustrate this situation, let us use the models of BS presented in Figure 7. In Figure 7(a) the baseline has all features implemented and in Figure 7(b) the best solution found is the most similar to the input models. Observe that in the best solution there exists an operation `withdrawWithoutLimit(amount: double)`. This operation is present in the variants that do not implement the feature WL (see Figure 1), i.e., it is present in three out of four variants. This operation is not present in the baseline model, so this baseline model does not provide a global overview of the variants. The baseline would not serve as reference for maintaining variants that do not have feature WL. However, in the architecture we can find out where the operation `withdrawWithoutLimit` is located.

3.4 Threats to Validity

The first threat to validity regards the parameter setting for the GA. We addressed this threat by adopting default values for crossover and mutation and set the values of population size and number of evaluations as big as possible. The second threat is the influence of the case studies. Despite of using only four case studies these systems are from different domains, and have different sizes. They can provide evidence about the usage of our approach. But nonetheless further studies should be conducted in the future. The third threat concerns the comparison to other approaches. To the best of our knowledge, there are no other studies with the exact same focus as ours. As baseline we used models known in advance that implement all features that compose the systems, assuming these models are the closest to an ideal solution.

4 Related Work

A detailed study on comparative techniques to architecture recovery is presented by Garcia et al. [7]. The authors observed that most techniques identify software components by using structural information from source code and do not present any technique based on UML models to recovery system architecture.

Hussain et al. apply a search-based technique to recovery software architecture using Particle Swarm Optimization that clusters system units based on cohesion and coupling of the source code [10]. , while Jeet and Dhir use a Genetic Black Hole to also perform clustering in the source code considering dependencies between system units [11]. Differently from our approach, none of them include UML models in the evolutionary process.

A search-based model merge approach is presented by Debreceni et al. who support collaborative model-driven engineering by merging models developed by different collaborators [3]. They propose a guided rule-based design space exploration where candidate models are generated to reach a conflict-free merged model. This approach also performs comparison in model states (state-based approach). However, it applies only a three-way model merge. In contrast, in our study we deal with the merging of multiple models. Kessentini et al. propose a search-based technique to merge models based on sequences of operations that originate different models [13, 16]. Operation-based merge considers the operations that perform modifications in a model, instead of the state of the model. Their goal is to find a sequence of operations to generate a merged model in order to minimize the number of conflicts and maximize the number of successfully applied operations. In both pieces of work the authors apply only three-way model merge. Our work differs from theirs in two points. First, we consider the state of the models (state-base) instead of the operations used to generate each variants (operation-based). Second, we deal with more than three models at once.

Maazoun et al. propose an approach to construct an SPL design from a set of class diagrams that are merged and then enriched with information from a Feature Model [15]. Martinez et al. create a model-based SPL by discovering variabilities and commonalities from model variants, which are then described using

Common Variability Language [17]. Even though they deal with model variants, in contrast with us their focus on SPLs imply another upfront investment to implement the benefits of systematic reuse, which is outside of our scope.

Rubin and Chechik propose an algorithm, named *NwM*, to merge multiple models simultaneously [21]. Their algorithm starts from a common set of model elements, the most frequent in the variants, and analyses all possible combination of remaining elements among the variants to find the best merging operations. This process is a polynomial-time approximation, since the problem is NP-Hard. It works for a limited number of models. Our approach differs from *NwM* because we do not need to identify the initial set of common elements and our search-based approach can deal with many model variants.

5 Conclusions

We presented in this paper an approach to discover model-based software architecture by merging UML model variants. Our approach relies on a search-based technique that does not require information regarding domain constrains or conflicting models elements in advance. The candidate architectures are evaluated by a measure called Model Similarity.

To evaluate our approach we performed an experiment with four case studies from different domains and with different sizes. The results show that our approach is able to find good candidate architectures even when features are implemented in multiple variants. Furthermore, we could observe that having a variant that implements all features of a system does not imply that this variant has all model elements spread in other variants.

We acknowledge that some results could be influenced by internal aspects of the case studies, however our approach is an easy way to support the discovery of a documented architecture. This architecture helps maintenance by (i) providing a global view of a set of variants that supports the identification of bad smells and refactoring activities; (ii) allowing reconciling design of different variants (potentially inconsistent) implemented by many designers; (iii) when a bug is fixed in one variant, the architecture helps to replicate the changes to other variants that also have the same model elements. The documented architecture supports evolution by (i) being a starting point to combine variants into an SPL, and (ii) reducing the time to produce variants with new combination of features.

For future work we plan to improve the match policy to include more detailed information regarding the semantics of the model variants. Furthermore, we want to evaluate our approach with more case studies to infer how model elements, i.e. implementation aspects, can have influence on getting good architectures.

References

1. Assunção, W.K.G., Lopez-Herrejon, R.E., Linsbauer, L., Vergilio, S.R., Egyed, A.: Reengineering legacy applications into software product lines: a systematic mapping. Empirical Software Engineering pp. 1–45 (2017)

2. Assunção, W.K.G., Lopez-Herrejon, R.E., Linsbauer, L., Vergilio, S.R., Egyed, A.: Extracting variability-safe feature models from source code dependencies in system variants. In: Genetic and Evolutionary Computation Conference (GECCO). pp. 1303–1310. ACM (2015)
3. Debreceni, C., Ráth, I., Varró, D., Carlos, X., Mendialdua, X., Trujillo, S.: International Conference Fundamental Approaches to Software Engineering (FASE), chap. Automated Model Merge by Design Space Exploration, pp. 104–121 (2016)
4. Dobrica, L., Niemela, E.: A survey on software architecture analysis methods. IEEE Transactions on Software Engineering 28(7), 638–653 (2002)
5. Durillo, J.J., Nebro, A.J.: jmetal: A java framework for multi-objective optimization. Advances in Engineering Software 42, 760–771 (2011), `http://jmetal.sourceforge.net/`
6. Faust, D., Verhoef, C.: Software product line migration and deployment. Software: Practice and Experience 33(10), 933–955 (2003)
7. Garcia, J., Ivkovic, I., Medvidovic, N.: A comparative analysis of software architecture recovery techniques. In: International Conference on Automated Software Engineering (ASE). pp. 486–496. IEEE (2013)
8. Goldberg, D.E., Deb, K., Clark, J.H.: Genetic algorithms, noise, and the sizing of populations. Complex Systems 6, 333–362 (1992)
9. Harman, M., Mansouri, S.A., Zhang, Y.: Search-based software engineering: Trends, techniques and applications. ACM Computing Surveys 45(1), 1–61 (2012)
10. Hussain, I., Khanum, A., Abbasi, A.Q., Javed, M.Y.: A novel approach for software architecture recovery using particle swarm optimization. The International Arab Journal of Information Technology 12(1), 32–41 (2015)
11. Jeet, K., Dhir, R.: Software architecture recovery using genetic black hole algorithm. ACM SIGSOFT Software Engineering Notes 40(1), 1–5 (2015)
12. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-Oriented Domain Analysis (FODA) feasibility study. Tech. rep., SEI - CMU (1990)
13. Kessentini, M., Werda, W., Langer, P., Wimmer, M.: Search-based model merging. In: Genetic and Evolutionary Computation Conference. pp. 1453–1460 (2013)
14. Krueger, C.W.: Software reuse. ACM Computing Surveys 24(2), 131–183 (1992)
15. Maazoun, J., Bouassida, N., Ben-Abdallah, H.: A bottom up spl design method. In: 2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD). pp. 309–316 (Jan 2014)
16. Mansoor, U., Kessentini, M., Langer, P., Wimmer, M., Bechikh, S., Deb, K.: Momm: Multi-objective model merging. Journal of Systems and Software 103, 423–439 (2015)
17. Martinez, J., Ziadi, T., Bissyandé, T.F., Klein, J., l. Traon, Y.: Automating the extraction of model-based software product lines from model variants. In: International Conference on Automated Software Engineering (ASE). pp. 396–406 (2015)
18. Martinez, J., Ziadi, T., Klein, J., Traon, Y.L.: Identifying and visualising commonality and variability in model variants. In: 10th European Conference Modelling Foundations and Applications (ECMFA). pp. 117–131 (2014)
19. Pohl, K., Böckle, G., van Der Linden, F.J.: Software product line engineering: foundations, principles and techniques. Springer Science & Business Media (2005)
20. Riva, C., Del Rosso, C.: Experiences with software product family evolution. In: International Workshop on Principles of Software Evolution. pp. 161–169 (2003)
21. Rubin, J., Chechik, M.: N-way model merging. In: 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE). pp. 301–311. ACM (2013)
22. Steinberg, D., Budinsky, F., Merks, E., Paternostro, M.: EMF: eclipse modeling framework. Pearson Education (2008)

Appendix D

Case Studies Details

The tables in the following present details of each variant of the case studies. In each table we present the name of the variants, the features and the number of model elements of each variant. Besides, we also indicate or present the baseline variants used in the experiment.

Table D.1: Banking System (BS)

Variant	Features				Model Elements				
	BS	WL	CON	CC	Cl	Int	Attr	Op	Rel
BS_P01	✓				3	0	5	6	1
BS_P02	✓	✓			3	0	6	8	1
BS_P03	✓		✓		4	0	6	7	3
BS_P04	✓			✓	4	0	7	11	3
BS_P05	✓		✓	✓	5	0	8	12	5
BS_P06	✓	✓		✓	4	0	8	13	3
BS_P07	✓	✓	✓		4	0	7	9	3
BS_P08*	✓	✓	✓	✓	5	0	9	14	4

BS: Base, WL: Withdraw Limit, CON: Consortium, CC: Currency Converter
* Baseline variant

Table D.2: Draw Product Line (DPL)

Variant	Features						Model Elements				
	DPL	L	R	C	W	F	Cl	Int	Attr	Op	Rel
DPL_P01	✓	✓					4	0	3	26	4
DPL_P02	✓	✓	✓				5	0	12	37	6
DPL_P03	✓		✓				4	0	11	29	4
DPL_P04	✓	✓	✓	✓			5	0	17	38	6
DPL_P05	✓	✓		✓			4	0	8	27	4
DPL_P06	✓		✓	✓			4	0	16	30	4
DPL_P07	✓	✓			✓		4	0	4	27	4
DPL_P08	✓	✓	✓		✓		5	0	13	38	6
DPL_P09	✓		✓		✓		4	0	12	30	4
DPL_P10	✓	✓	✓	✓	✓		5	0	18	39	6
DPL_P11	✓	✓		✓	✓		9	0	9	69	10
DPL_P12	✓		✓	✓	✓		4	0	17	31	4
DPL_P13	✓	✓	✓	✓		✓	5	0	19	40	6
DPL_P14	✓		✓	✓		✓	4	0	18	32	4
DPL_P15*	✓	✓	✓	✓	✓	✓	5	0	20	41	6
DPL_P16	✓		✓	✓	✓	✓	4	0	19	33	4

DPL: Base, L: Line, R: Rectangle, C: Color, W: Wipe, F: Fill
* Baseline variant

Table D.3: Mobile Media - Version 1 (MMv1)

Variant	Features					Model Elements				
	MM	IP	132x176	128x149	176x205	Cl	Int	Attr	Op	Rel
MMv1_P01*	✓	✓	✓			23	1	22	121	13
MMv1_P02*	✓	✓		✓		23	1	22	121	13
MMv1_P03*	✓	✓			✓	23	1	22	121	13

MM: MobileMedia, IP includePhoto, 132x176: device_screen_132x176,
128x149: device_screen_128x149, 176x205: device_screen_176x205
* Baseline variant

Table D.4: Video On Demand (VOD)

Variant	Features											Model Elements				
	VOD	SP	SelectM	StartM	PI	VRC	P	StopM	QP	CS	D	Cl	Int	Attr	Op	Rel
VOD_P00	✓	✓	✓	✓	✓	✓						32	0	280	217	79
VOD_P01	✓	✓	✓	✓	✓	✓	✓					32	0	280	217	79
VOD_P02	✓	✓	✓	✓	✓	✓		✓				33	0	280	221	81
VOD_P03	✓	✓	✓	✓	✓	✓	✓	✓				33	0	280	221	81
VOD_P04	✓	✓	✓	✓	✓	✓			✓			33	0	280	221	81
VOD_P05	✓	✓	✓	✓	✓	✓	✓		✓			33	0	280	221	81
VOD_P06	✓	✓	✓	✓	✓	✓		✓	✓			34	0	280	225	83
VOD_P07	✓	✓	✓	✓	✓	✓	✓	✓	✓			34	0	280	225	83
VOD_P08	✓	✓	✓	✓	✓	✓				✓		37	0	280	232	92
VOD_P09	✓	✓	✓	✓	✓	✓	✓			✓		37	0	280	232	92
VOD_P10	✓	✓	✓	✓	✓	✓		✓		✓		38	0	280	236	94
VOD_P11	✓	✓	✓	✓	✓	✓	✓	✓		✓		38	0	280	236	94
VOD_P12	✓	✓	✓	✓	✓	✓			✓	✓		38	0	280	236	94
VOD_P13	✓	✓	✓	✓	✓	✓	✓		✓	✓		38	0	280	236	94
VOD_P14	✓	✓	✓	✓	✓	✓		✓	✓	✓		39	0	280	240	96
VOD_P15	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		39	0	280	240	96
VOD_P16	✓	✓	✓	✓	✓	✓					✓	35	0	280	226	87
VOD_P17	✓	✓	✓	✓	✓	✓	✓				✓	35	0	280	226	87
VOD_P18	✓	✓	✓	✓	✓	✓		✓			✓	36	0	280	230	89
VOD_P19	✓	✓	✓	✓	✓	✓	✓	✓			✓	36	0	280	230	89
VOD_P20	✓	✓	✓	✓	✓	✓			✓		✓	36	0	280	230	89
VOD_P21	✓	✓	✓	✓	✓	✓	✓		✓		✓	36	0	280	230	89
VOD_P22	✓	✓	✓	✓	✓	✓		✓	✓		✓	37	0	280	234	91
VOD_P23	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	37	0	280	234	91
VOD_P24	✓	✓	✓	✓	✓	✓				✓	✓	40	0	280	241	100
VOD_P25	✓	✓	✓	✓	✓	✓	✓			✓	✓	40	0	280	241	100
VOD_P26	✓	✓	✓	✓	✓	✓		✓		✓	✓	41	0	280	245	102
VOD_P27	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	41	0	280	245	102
VOD_P28	✓	✓	✓	✓	✓	✓			✓	✓	✓	41	0	280	245	102
VOD_P29	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	41	0	280	245	102
VOD_P30	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	42	0	280	249	104
VOD_P31*	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	42	0	280	249	104

VOD: Base, SP: StartPlayer, SelectM: SelectMovie, StartM: StartMovie, PI: PlayImm, VRC: VRCInterface, P: Pause, StopM: StopMovie, QP: QuitPlayer, CS: ChangeServer, D: Detail

* Baseline variant

Table D.5: ZipMe (ZM)

Variant	Features							Model Elements				
	ZM	C	CRC	AC	GZIP	Adler32	E	Cl	Int	Attr	Op	Rel
ZipMe_P00	✓	✓						22	3	183	241	64
ZipMe_P01	✓	✓	✓					23	3	185	251	66
ZipMe_P02	✓	✓		✓				22	3	183	243	66
ZipMe_P03	✓	✓	✓	✓				23	3	185	253	68
ZipMe_P04	✓	✓			✓			25	3	193	263	68
ZipMe_P05	✓	✓	✓		✓			26	3	195	282	73
ZipMe_P06	✓	✓		✓	✓			25	3	193	265	70
ZipMe_P07	✓	✓	✓	✓	✓			26	3	195	284	75
ZipMe_P08	✓	✓				✓		23	3	185	263	69
ZipMe_P09	✓	✓	✓			✓		24	3	187	273	71
ZipMe_P10	✓	✓		✓		✓		23	3	185	265	71
ZipMe_P11	✓	✓	✓	✓		✓		24	3	187	275	73
ZipMe_P12	✓	✓			✓	✓		26	3	195	285	73
ZipMe_P13	✓	✓	✓		✓	✓		27	3	197	304	78
ZipMe_P14	✓	✓		✓	✓	✓		26	3	195	287	75
ZipMe_P15	✓	✓	✓	✓	✓	✓		27	3	197	306	80
ZipMe_P16	✓	✓					✓	23	3	189	262	70
ZipMe_P17	✓	✓	✓				✓	24	3	191	279	74
ZipMe_P18	✓	✓		✓			✓	23	3	189	264	72
ZipMe_P19	✓	✓	✓	✓			✓	24	3	191	281	76
ZipMe_P20	✓	✓			✓		✓	26	3	199	284	74
ZipMe_P21	✓	✓	✓		✓		✓	27	3	201	310	81
ZipMe_P22	✓	✓		✓	✓		✓	26	3	199	286	76
ZipMe_P23	✓	✓	✓	✓	✓		✓	27	3	201	312	83
ZipMe_P24	✓	✓				✓	✓	24	3	191	284	75
ZipMe_P25	✓	✓	✓			✓	✓	25	3	193	301	79
ZipMe_P26	✓	✓		✓		✓	✓	24	3	191	286	77
ZipMe_P27	✓	✓	✓	✓		✓	✓	25	3	193	303	81
ZipMe_P28	✓	✓			✓	✓	✓	27	3	201	306	79
ZipMe_P29	✓	✓	✓		✓	✓	✓	28	3	203	332	86
ZipMe_P30	✓	✓		✓	✓	✓	✓	27	3	201	308	81
ZipMe_P31*	✓	✓	✓	✓	✓	✓	✓	28	3	203	334	88

ZM: Base, C: Compress, CRC: CRC, AC: ArchiveCheck, GZIP: GZIP, Adler32: Adler32Checksum, E: Extract
* Baseline variant

Table D.6: Game Of Life (GOL)

Variant	Features															Model Elements				
	GB	MB	AG	RDG	RG	FG	FDG	GS	T	IO	PM	URGB	URG	UR	URT	Cl	Int	Attr	Op	Rel
GOL_P00	✓	✓														12	1	19	57	29
GOL_P01	✓	✓	✓	✓	✓											14	2	19	69	31
GOL_P02	✓	✓	✓			✓	✓									14	2	21	69	31
GOL_P03	✓	✓	✓	✓	✓	✓		✓								16	2	21	73	33
GOL_P04	✓	✓	✓		✓	✓	✓	✓								16	2	21	73	33
GOL_P05	✓	✓	✓	✓	✓				✓							15	2	19	75	31
GOL_P06	✓	✓	✓			✓	✓		✓							15	2	21	75	31
GOL_P07	✓	✓	✓	✓	✓	✓		✓	✓							17	2	21	79	33
GOL_P08	✓	✓	✓		✓	✓	✓	✓	✓							17	2	21	79	33
GOL_P09	✓	✓	✓	✓	✓					✓						15	2	24	72	31
GOL_P10	✓	✓	✓			✓	✓			✓						15	2	26	72	31
GOL_P11	✓	✓	✓	✓	✓	✓		✓		✓						17	2	26	76	33
GOL_P12	✓	✓	✓		✓	✓	✓	✓		✓						17	2	26	76	33
GOL_P13	✓	✓	✓	✓	✓				✓	✓						16	2	24	78	31
GOL_P14	✓	✓	✓			✓	✓		✓	✓						16	2	26	78	31
GOL_P17	✓	✓	✓	✓	✓						✓					15	2	21	70	33
GOL_P18	✓	✓	✓			✓	✓				✓					15	2	23	70	33
GOL_P19	✓	✓	✓	✓	✓	✓		✓			✓					17	2	23	74	35
GOL_P20	✓	✓	✓		✓	✓	✓	✓			✓					17	2	23	74	35
GOL_P21	✓	✓	✓	✓	✓				✓		✓					16	2	21	76	33
GOL_P22	✓	✓	✓			✓	✓		✓		✓					16	2	23	76	33
GOL_P25	✓	✓	✓	✓	✓					✓	✓					16	2	26	73	33
GOL_P26	✓	✓	✓			✓	✓			✓	✓					16	2	28	73	33
GOL_P29	✓	✓	✓	✓	✓				✓	✓	✓					17	2	26	79	33
GOL_P30	✓	✓	✓			✓	✓		✓	✓	✓					17	2	28	79	33
GOL_P33	✓	✓	✓	✓	✓							✓	✓	✓		14	2	19	79	32
GOL_P34	✓	✓	✓			✓	✓					✓	✓	✓		14	2	21	79	32
GOL_P50	✓	✓	✓			✓	✓		✓			✓	✓	✓	✓	15	2	21	85	32
Baselines																				
GOL_P63	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	19	2	28	93	36
GOL_P64	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	19	2	28	93	36

GB: GuiBase, MB: ModelBase, AG: AbstractGenerator, RDG: RandomDefaultGenerator, RG: RandomGenerator, FG: FormGenerator, FDG: FormDefaultGenerator, GS: GeneratorSelection, T: Test, IO: IO, PM: PopUpMenu, URGB: UndoRedoGuiBase, URG: UndoRedoGenerator, UR: UndoRedo, URT UndoRedoTest

Table D.7: Mobile Media - Version 2 (MMv2)

Variant	Features						Model Elements				
	MM	IP	132x176	128x149	176x205	IS	Cl	Int	Attr	Op	Rel
MMv2_P01	✓	✓	✓				24	1	25	133	21
MMv2_P02	✓	✓		✓			24	1	25	133	21
MMv2_P03	✓	✓			✓		24	1	25	133	21
MMv2_P04	✓	✓	✓			✓	24	1	26	138	21
Baselines											
MMv2_P04	✓	✓	✓			✓	24	1	26	138	21
MMv2_P04	✓	✓		✓		✓	24	1	26	138	21
MMv2_P04	✓	✓			✓	✓	24	1	26	138	21

MM: MobileMedia, IP: includePhoto, 132x176: device_screen_132x176, 128x149: device_screen_128x149, 176x205: device_screen_176x205, IS: includeSorting

Table D.8: Mobile Media - Version 3 (MMv3)

Variant	Features							Model Elements				
	MM	IP	132x176	128x149	176x205	IF	IS	Cl	Int	Attr	Op	Rel
MMv3_P01	✓	✓	✓					24	1	25	133	21
MMv3_P02	✓	✓		✓				24	1	25	133	21
MMv3_P03	✓	✓			✓			24	1	25	133	21
MMv3_P04	✓	✓	✓			✓		24	1	26	136	21
MMv3_P05	✓	✓		✓		✓		24	1	26	136	21
MMv3_P06	✓	✓			✓	✓		24	1	26	136	21
MMv3_P07	✓	✓	✓				✓	24	1	26	138	21
MMv3_P08	✓	✓		✓			✓	24	1	26	138	21
MMv3_P09	✓	✓			✓		✓	24	1	26	138	21
Baselines												
MMv3_P10	✓	✓	✓			✓	✓	24	1	27	141	21
MMv3_P11	✓	✓		✓		✓	✓	24	1	27	141	21
MMv3_P12	✓	✓			✓	✓	✓	24	1	27	141	21

MM: MobileMedia, IP: includePhoto, 132x176: device_screen_132x176, 128x149: device_screen_128x149, 176x205: device_screen_176x205, IF: includeFavourites, IS: includeSorting

Table D.9: Mobile Media - Version 4 (MMv4)

Variant	Features								Model Elements				
	MM	IP	132x176	128x149	176x205	ICM	IF	IS	Cl	Int	Attr	Op	Rel
MMv4_P01	✓	✓	✓						28	1	25	147	25
MMv4_P02	✓	✓		✓					28	1	25	147	25
MMv4_P03	✓	✓			✓				28	1	25	147	25
MMv4_P04	✓	✓	✓			✓			29	1	26	151	28
MMv4_P05	✓	✓		✓		✓			29	1	26	151	28
MMv4_P06	✓	✓			✓	✓			29	1	26	151	28
MMv4_P07	✓	✓	✓				✓		28	1	26	150	25
MMv4_P08	✓	✓		✓			✓		28	1	26	150	25
MMv4_P09	✓	✓			✓		✓		28	1	26	150	25
MMv4_P13	✓	✓	✓					✓	28	1	26	152	25
MMv4_P14	✓	✓		✓				✓	28	1	26	152	25
MMv4_P15	✓	✓			✓			✓	28	1	26	152	25
Baselines													
MMv4_P22	✓	✓	✓			✓	✓	✓	29	1	28	159	28
MMv4_P23	✓	✓		✓		✓	✓	✓	29	1	28	159	28
MMv4_P24	✓	✓			✓	✓	✓	✓	29	1	28	159	28

MM: MobileMedia, IP: includePhoto, 132x176: device_screen_132x176, 128x149: device_screen_128x149, 176x205: device_screen_176x205, ICM: includeCopyMedia, IF: includeFavourites, IS: includeSorting

Table D.10: Mobile Media - Version 5 (MMv5)

Variant	Features									Model Elements				
	MM	IP	132x176	128x149	176x205	SMS	ICM	IF	IS	Cl	Int	Attr	Op	Rel
MMv5_P01	✓	✓	✓							28	1	25	147	25
MMv5_P02	✓	✓		✓						28	1	25	147	25
MMv5_P03	✓	✓			✓					28	1	25	147	25
MMv5_P04	✓	✓	✓			✓				36	1	43	190	29
MMv5_P05	✓	✓		✓		✓				36	1	43	190	29
MMv5_P06	✓	✓			✓	✓				36	1	43	190	29
MMv5_P07	✓	✓	✓				✓			29	1	26	154	28
MMv5_P08	✓	✓		✓			✓			29	1	26	154	28
MMv5_P09	✓	✓			✓		✓			29	1	26	154	28
MMv5_P10	✓	✓	✓			✓	✓			36	1	43	192	31
MMv5_P11	✓	✓		✓		✓	✓			36	1	43	192	31
MMv5_P12	✓	✓			✓	✓	✓			36	1	43	192	31
MMv5_P13	✓	✓	✓					✓		28	1	26	150	25
MMv5_P14	✓	✓		✓				✓		28	1	26	150	25
MMv5_P15	✓	✓			✓			✓		28	1	26	150	25
MMv5_P16	✓	✓	✓			✓		✓		36	1	44	193	29
MMv5_P17	✓	✓		✓		✓		✓		36	1	44	193	29
MMv5_P18	✓	✓			✓	✓		✓		36	1	44	193	29
MMv5_P19	✓	✓	✓				✓	✓		29	1	27	157	28
MMv5_P20	✓	✓		✓			✓	✓		29	1	27	157	28
MMv5_P21	✓	✓			✓		✓	✓		29	1	27	157	28
MMv5_P25	✓	✓	✓						✓	28	1	26	152	25
MMv5_P26	✓	✓		✓					✓	28	1	26	152	25
MMv5_P27	✓	✓			✓				✓	28	1	26	152	25
MMv5_P28	✓	✓	✓			✓			✓	36	1	44	195	29
MMv5_P29	✓	✓		✓		✓			✓	36	1	44	195	29
MMv5_P30	✓	✓			✓	✓			✓	36	1	44	195	29
MMv5_P31	✓	✓	✓				✓		✓	29	1	27	159	28
MMv5_P32	✓	✓		✓			✓		✓	29	1	27	159	28
MMv5_P33	✓	✓			✓		✓		✓	29	1	27	159	28
MMv5_P37	✓	✓	✓					✓	✓	28	1	27	155	25
MMv5_P38	✓	✓		✓				✓	✓	28	1	27	155	25
MMv5_P39	✓	✓			✓			✓	✓	28	1	27	155	25
Baselines														
MMv5_P46	✓	✓	✓			✓	✓	✓	✓	36	1	45	200	31
MMv5_P47	✓	✓		✓		✓	✓	✓	✓	36	1	45	200	31
MMv5_P48	✓	✓			✓	✓	✓	✓	✓	36	1	45	200	31

MM: MobileMedia, IP: includePhoto, 132x176: device_screen_132x176, 128x149: device_screen_128x149, 176x205: device_screen_176x205, SMS: includeSmsFeature, ICM: includeCopyMedia, IF: includeFavourites, IS: includeSorting